INFORMATION AND COMPUTATION **126**, 123–143 (1996)
ARTICLE NO. 0041

ROEL BLOO

Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, the Netherlands
E-mail: bloo@win.tue.nl

FAIROUZ KAMAREDDINE

Department of Computing Science, University of Glasgow, 17 Lilybank Gardens, Glasgow G12 8QQ, Scotland
E-mail: fairouz@dcs.glasgow.ac.uk

AND

ROB NEDERPELT

Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, the Netherlands
E-mail: wsinrpn@win.tue.nl

In this paper, we propose to extend the Barendregt Cube by generalising β-reduction and by adding definition mechanisms. Generalised reduction allows contracting more visible redexes than usual, and definitions are an important tool to allow for a more flexible typing system. We show that this extension satisfies most of the original properties of the Cube including Church-Rosser, Subject Reduction and Strong Normalisation.

1. INTRODUCTION

1.1. *Why Generalised Reduction*

The usual notion of reduction in the λ-calculus might not be as general as one desires, as the following example shows (we ignore types for the sake of clarity):

EXAMPLE 1.1. In $A \equiv ((\lambda_f.(\lambda_x.\lambda_y.fxy)m)+)n$, we have the redexes $(\lambda_x.\lambda_y.fxy)m$ and $(\lambda_f.(\lambda_x.\lambda_y.fxy)m)+$. There is however a virtual redex which is not immediately visible in the classical term; namely, $(\lambda_y.+my)n$. Such a redex will only be visible after we have contracted the above two redexes and can be said to arise in the computation. Furthermore, one may want to contract the redex based on $(\lambda_y.-)n$ (resulting in the term $(\lambda_f.(\lambda_x.fxn)m)+$) before one has contracted either of the redexes $(\lambda_f.-)+$ and $(\lambda_x.-)m$.

All the above three redexes are *needed* to reach the normal form of A. The virtual redex, however, could only be seen once we had contracted the first two redexes. There is, moreover, a wish to make as many needed redexes as possible visible and even though the notion of a needed redex is undecidable, much work has been carried out in order to study some classes of needed redexes (as in [BKKS87, Gardner94, BKN9-]). Our proposal is not only to make as many redexes as possible visible, but also to give newly visible redexes the possibility of being contracted before other ones.

First, this view on reduction gives an appropriate tool for the study of some programming languages. For example, in lazy evaluation [Launchbury93], some redexes get frozen while other ones are being contracted. Now, if we had the ability to choose which redex to contract out of all visible redexes, rather than waiting for some redex to be evaluated first, then we could say that we had achieved a flexible system where we had control over what to contract rather than letting reductions force themselves in some order. Second, we think that an investigation concerning the *complete* class of visible redexes in a term gives a better understanding of reduction strategies, e.g., the optimal reductions as in [Lévy80].

* The authors were supported by the Netherlands Computer Science Research Foundation (SION, NWO), the Basic Action for Research ESPRIT project "Types for Proofs and Programs," and the ESPSRC Grant GR/K 25014 and are grateful to the Department of Computing Science, Glasgow University, and to the Department of Mathematics and Computing Science, Eindhoven University of Technology, for their financial support and hospitality. We thank Henk Barendregt, Bob Constable, Herman Geuvers, Stefan Khars, Tom Melham, and Joe Wells for their useful discussions and remarks.

0890-5401/96 $18.00

1.2. *Why Definition Mechanisms*

Practical experiences with type systems show that definitions are indispensable for any realistic application. Without definitions, terms soon become forbiddingly complicated. By using definitions one can avoid such an explosion in complexity. This is, by the way, a very natural thing to do: the apparatus of mathematics, for instance, is unimaginable without definitions.

In many type theories and lambda calculi, there is no possibility of introducing definitions. This possibility is essential for practical use, and indeed implementations of Pure Type Systems such as Nuprl [CON86], Coq [Dow91], Lego [LP92] and HOL [GM93] do provide this possibility. Moreover, experience with Automath [NGV94] has shown the need for definitions. But what are definitions and why are they attractive? Definitions are name abbreviating expressions and occur in contexts where we reason about terms.

EXAMPLE 1.2. $\texttt{Let } id{=}(\lambda_{x:A}.x){:}A{\to}A \texttt{ in } (\lambda_{y:A\to A}.id)\ id$ defines *id* to be $(\lambda_{x:A}.x)$ in a more complex expression in which *id* occurs two times.

The intended meaning of a definition $x = a$ is that the definiendum x can be substituted by the definiens a in some expression b. In a sense, an expression $\texttt{let } x : A \texttt{ be } a \texttt{ in } b$ is similar to $(\lambda_{x:A}.b)a$. It is not intended, however, to substitute all the occurrences of x in b by a. Nor is it intended that such a definition be a part of our term. Rather, the definition will live in the environment (or context) in which we evaluate or reason about the expression.

One of the advantages of the definition $\texttt{let } x : A \texttt{ be } a \texttt{ in } b$ over $(\lambda_{x:A}.b)a$ is that it is convenient to have the freedom of substituting only some of the occurrences of an expression in a given formula. Another advantage is efficiency; one evaluates a in $\texttt{let } x : A \texttt{ be } a \texttt{ in } b$ only once, even in lazy languages. A further advantage is that defining x to be a in b can be used to type b efficiently, since the type A of a has to be calculated only once. Moreover, a definition may be necessary to type a term as is shown in the following example.

EXAMPLE 1.3. Without definitions, it is not possible to type $\lambda_{y:x}.\lambda_{f:a\to a}.fy$ even when we somehow know that x is an abbreviation for a. This is because f expects an argument of type a, and y is of type x. Once we make use of the fact that x is defined to be a in our context, then y will have type a and the term will be typable, as we see in Example 1.6.

Introducing definitions in Pure Type Systems is an interesting subject of research at the moment. For example, [SP93] extended PTSs with definitions. Our approach allows such an extension to be made in an elegant way. In fact, the generated type derivations for terms in the Cube with definitions become much shorter than those in the absence of definitions (see Section 7.2). Moreover, we do not have to use complex relations to introduce definitions as in [SP93]. Rather, the extension will be a natural way to how our terms are written. Basic for our proposed extensions is a new notation: *the item notation.*

1.3. *The Item Notation for Definitions and Generalised Reduction*

The *item notation* is a simple variant of the usual notation where the argument is given before the function, the type is given before the abstraction operator, and where the parentheses are grouped differently than those of the classical notation. So that, if $\mathscr{I}$ translates classical terms into our notation, then $\mathscr{I}(AB)$ is written as $(\mathscr{I}(B)\delta)\ \mathscr{I}(A)$ (here, δ is a special symbol used for application) and $\mathscr{I}(\mathscr{O}_{x:A}.B)$ is written as $(\mathscr{I}(A)\ \mathscr{O}_x)\ \mathscr{I}(B)$ where $\mathscr{O} = \lambda$ or Π. Both $(t\delta)$ and $(t\mathscr{O}_x)$, t being a term in item notation, are called *items*. For reasons explaining the usefulness of such a notation, the reader is referred to [KN93, KN96a]. For this paper however, the reader is to notice that redexes and definitions can be easily generalised and introduced with item notation. A traditional redex is a term that starts with a δ-item next to a λ-item. A definition is itself a certain form of a δ-item next to a λ-item.

EXAMPLE 1.4. $\mathscr{I}((\lambda_{x:A\to(B\to C)}.\lambda_{y:A}.xy)\,t) \equiv (t\delta)(A\to(B\to C)\lambda_x)(A\lambda_y)(y\delta)x$. The items are $(t\delta)$, $(A\to(B\to C)\lambda_x)$, $(A\lambda_y)$, and $(y\delta)$. The definition is $(t\delta)(A\to(B\to C)\ \lambda_x)$ and the redex is the whole term.

DEFINITION 1.5 (Classical Redexes and β-Reduction in Item Notation). In the item notation of the λ-calculus, a classical redex $(\lambda_{x:B}.A)C$ is of the form $(C\delta)(B\lambda_x)A$. We call the pair $(C\delta)(B\lambda_x)$, a *$\delta\lambda$-pair*, or a *$\delta\lambda$-segment*. The β-reduction axiom (β) is: $(C\delta)(B\lambda_x)A \to_\beta A[x := C]$. One-step β-reduction $\to_\beta$ is the compatible relation generated out of (β). Many step β-reduction, $\twoheadrightarrow_\beta$, is the reflexive transitive closure of $\to_\beta$.

In item notation, the term A of Example 1.1 becomes $(n\delta)(+\delta)(\lambda_f)(m\delta)(\lambda_x)(\lambda_y)(y\delta)(x\delta)f$ (recall that we ignore types). The two classical redexes correspond to $\delta\lambda$-pairs as follows:

1. $(\lambda_x.\lambda_y.fxy)m$ corresponds to $(m\delta)(\lambda_x)$. The remainder of the redex, $(\lambda_y)(y\delta)(x\delta)f$, corresponds to the maximal subterm of A to the right of (λ_x).
2. $(\lambda_f.(\lambda_x.\lambda_y.fxy)m)+$ corresponds to $(+\delta)(\lambda_f)$, the rest being $(m\delta)(\lambda_x)(\lambda_y)(y\delta)(x\delta)f$.

Looking closely at A written in item notation, one sees that the third redex described in Example 1.1 is obtained by just matching δ and λ-items. $(\lambda_y.fxy)n$ is visible as it corresponds to the matching $(n\delta)(\lambda_y)$ where $(n\delta)$ and (λ_y) are separated by $(+\delta)(\lambda_f)(m\delta)(\lambda_x)$. Hence, by extending

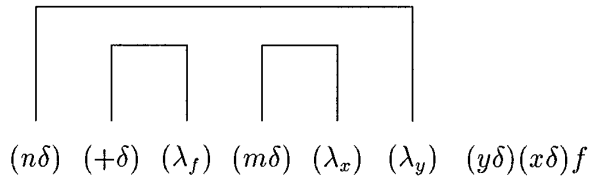

FIG. 1. Extended redexes in item notation.

the notion of a redex from being a δ-item adjacent to a λ-item, to being a matching pair of δ- and λ-items, we can make more redexes visible. Such an extension is simple, as in $(C\delta)\,\bar{s}(B\lambda_x)$, we say that $(C\delta)$ and $(B\lambda_x)$ match if $\bar{s}$ has the same structure as a matching composite of opening and closing brackets, each δ-item corresponding to an opening bracket and each λ-item corresponding to a closing bracket. For example, in A above, $(n\delta)$ and (λ_y) match as $(+\delta)(\lambda_f)(m\delta)(\lambda_x)$ has the bracketing structure $[\ ][\ ]$ (see Fig. 1). We refine β-reduction by changing (β) to

$$(C\delta)\,\bar{s}(B\lambda_x)\,A \hookrightarrow_\beta \bar{s}(A[x := C]) \text{ if } (C\delta) \text{ and } (B\lambda_x) \text{ match}$$

Now, what about definitions? The first step is to define definitions as matching $\delta\lambda$-couples and to include them in contexts with the condition that if a definition occurs in a context then it can be used anywhere in the term we are reasoning about in that context. Hence, if we look at Example 1.3, then we can type the term now that we allow definitions to occur in contexts and we extend $\vdash$ slightly so that it can see what is in its context.

EXAMPLE 1.6. We use as context the segment $(a\delta)(A\lambda_x)(x\lambda_y)(a \to a\lambda_f)$, establishing that x of type A is defined as a, that y has type x, and that f has type $a \to a$. Then, making use of this definition, we have

$$\begin{aligned}
&(a\delta)(A\lambda_x)(x\lambda_y)(a \to a\lambda_f) \vdash f: a \to a\\
&(a\delta)(A\lambda_x)(x\lambda_y)(a \to a\lambda_f) \vdash y: x = a\\
&(a\delta)(A\lambda_x)(x\lambda_y)(a \to a\lambda_f) \vdash (y\delta) f: a\\
&(a\delta)(A\lambda_x)(x\lambda_y) \vdash (a \to a\lambda_f)(y\delta) f: (a \to a) \to a\\
&(a\delta)(A\lambda_x) \vdash (x\lambda_y)(a \to a\lambda_f)(y\delta) f: x \to (a \to a) \to a\\
&\qquad = a \to (a \to a) \to a
\end{aligned}$$

Based on the above discussion, we divide the paper into the following sections:

- In Section 2, we introduce the item notation.
- In Section 3, we recall the Cube as in [Bar92], and all its properties.
- In Section 4, we add to the Cube generalised reduction $\hookrightarrow_\beta$ and show that $\hookrightarrow\!\!\!\!\rightarrow_\beta$ (the reflexive transitive closure of $\hookrightarrow_\beta$) generalises $\twoheadrightarrow_\beta$ (Lemma 4.3) such that $=_\beta$ and $\approx_\beta$ are the same (Lemma 4.5). This means that almost all the original properties still hold for $\hookrightarrow\!\!\!\!\rightarrow_\beta$. However,

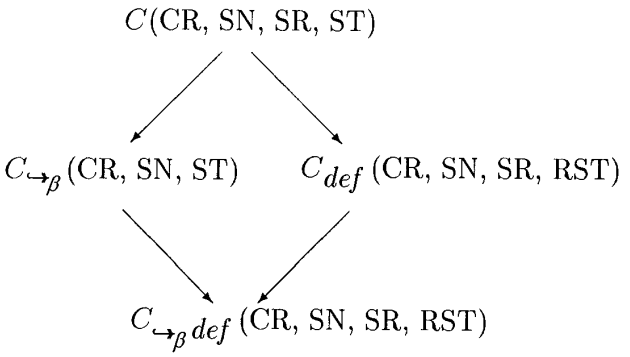

FIG. 2. Properties of the Cube with various extensions.

Church–Rosser (CR), Subject Reduction (SR), Subtyping (ST), and Strong Normalisation (SN) deserve special attention. CR, ST and SN are shown to hold (without the need for SR in the case of SN). SR holds in $\lambda\underline{\omega}$ and $\lambda_{\to}$, but fails in the remaining systems. This problem is solved in Section 6 by adding definitions.

- In Section 5 we add definitions to the Cube and show that all the properties of [Barg2] (including SR) hold with definitions, except ST. We show, however, that a restricted form of ST, RST, still holds. CR is not touched by the addition of definitions, contrary to the account of [SP93], where a reduction relation was introduced to capture definitions and hence CR had to be shown.
- In Section 6, we extend the Cube with both generalised reduction and definitions. We show that the Cube extended with definitions and generalised reduction preserves all its important properties, again except ST but we have RST. We present in particular, the general proof of Strong Normalisation which applies to all the earlier systems.
- In Section 7, we discuss the conservativity of the Cube with definitions, with respect to the Cube without definitions. We show that more terms are typable using definitions. However, when a judgment is derivable in a system of the Cube with definitions, the judgment itself where all the definitions are unfolded is derivable without definitions (Theorem 7.3). We also compare our system of definitions with that of [SP93] and discuss type checking and the length of derivations using definitions. Finally, we mention the relation of our work to that of Automath.

Figure 2 summarizes our results, showing that one can safely use the Cube with definitions only, or with both definitions and generalised reduction. When using generalised reduction without definitions, one must remain in the systems $\lambda_{\to}$ and $\lambda\underline{\omega}$ as the other systems lose their SR.

2. THE ITEM NOTATION

For a detailed description of item notation, the reader is referred to [KN93, KN94, KN95, KN96a]. We will

introduce in this section the minimum machinery needed to represent the Cube in item notation and for introducing generalised reduction and definitions.

The systems of the Cube are based on a set of *pseudo-expressions* $\mathscr{T}$ defined by

$$\mathscr{T} = V \mid C \mid (\mathscr{T}\delta)\mathscr{T} \mid (\mathscr{T}\mathscr{O}_V)\mathscr{T},$$

where V and C are infinite collections of variables and constants respectively. We assume that $x, y, z, \ldots$ range over V and we take two special constants $*$ and $\Box$. These constants are called sorts and the meta-variables $S, S_1, S_2, \ldots$ are used to range over the set of sorts $\mathscr{S} = \{*, \Box\}$. We take $A, B, C, a, b \ldots$ to range over pseudo-expressions. Parentheses will be omitted when no confusion occurs. For convenience's sake, we divide V into two disjoint sets V^* and $V^\Box$, the sets of object and constructor variables respectively. We take $x^*, y^*, z^*, \ldots$ to range over V^* and $x^\Box, y^\Box, z^\Box, \ldots$ to range over $V^\Box$. Throughout, we let $\mathscr{O}$ range over $\{\lambda, \Pi\}$.

Bound and free variables and substitution are defined as usual. We write $BV(A)$ and $FV(A)$ to represent the bound and free variables of A respectively. We write $A[x := B]$ to denote the term where all the free occurrences of x in A have been replaced by B. Furthermore, we take terms to be equivalent up to variable renaming and use $\equiv$ to denote syntactical equality of terms. We assume moreover, the Barendregt variable convention, BC, which says that names of bound variables differ from the free ones in a term and that different λ's have different variables as subscript. Now, some machinery for item notation follows.

DEFINITION 2.1 ((Main) Items, (Main, $\delta\mathscr{O}$-) Segments, End Variable, Weight).

- If x is a variable, and A is a pseudo-expression then $(A\lambda_x)$, $(A\Pi_x)$ and $(A\delta)$ are items (called λ-item, Π-item and δ-item respectively). We use $s, s_1, s_i, \ldots$ to range over items.

- A concatenation of zero or more items is a *segment*. We use $\bar{s}, \bar{s}_1, \bar{s}_i, \ldots$ to range over segments and write $\varnothing$ for the empty segment. A $\delta\mathscr{O}$-segment is a δ-item immediately followed by an $\mathscr{O}$-item. If $\bar{s} \equiv s_1 s_2 \cdots s_n$, we call $s_1, s_2, \ldots, s_n$, the *main items* of $\bar{s}$.

- Each pseudo-expression A is the concatenation of zero or more items and a variable or constant: $A \equiv s_1 s_2 \cdots s_n x$. These items $s_1, s_2, \ldots, s_n$ are called the *main items* of A; a concatenation of adjacent main items $s_m \cdots s_{m+k}$, is called a *main segment* of A.

- The *weight* of a segment $\bar{s}$, $\mathtt{weight}(\bar{s})$, is the number of main items that compose the segment. Moreover, we define $\mathtt{weight}(\bar{s}x) = \mathtt{weight}(\bar{s})$.

DEFINITION 2.2 (Well-Balanced Segments, Definitions, Definition Unfolding).

- $\varnothing$ is a well-balanced segment.

- If $\overline{s_1}$, $\overline{s_2}$ are well-balanced, then $(A\delta)\ \overline{s_1}(B\mathscr{O}_x)\ \overline{s_2}$ is well-balanced.

- If $\bar{s}$ is well-balanced and does not contain main Π-items, then $(A\delta)\ \bar{s}(B\lambda_x)$ occurring in a context is called a definition.

- Let $\bar{s}$ be a well-balanced segment, occurring in a context, which consists of definitions, and let $A \in \mathscr{T}$. We define the unfolding of the definitions of $\bar{s}$ in A, written $[A]_{\bar{s}}$, inductively as follows: $[A]_{\varnothing} \equiv A$, $[A]_{(B\delta)\,\overline{s_1}(C\lambda_x)} \equiv [A[x := B]]_{\overline{s_1}}$ and $[A]_{\overline{s_1}\,\overline{s_2}} \equiv [[A]_{\overline{s_2}}]_{\overline{s_1}}$. Note that substitution takes place from right to left and that when none of the binding variables of $\bar{s}$ are free in A, then $[A]_{\bar{s}} \equiv A$.

Remark 2.3. We maintain the same liberal attitude for definitions as we did for generalised redexes. That is, not only $(A\delta)(B\lambda_x)$ may act as a definition in a context, but also $(A\delta)\ \bar{s}(B\lambda_x)$ for any well-balanced segment $\bar{s}$ without main Π-items.

Note that we speak of definitions when such an $(A\delta)\ \bar{s}(B\lambda_x)$ occurs in a context; otherwise, when $(A\delta)\ \bar{s}(B\lambda_x)$ occurs in a term, we speak of a β-redex.

DEFINITION 2.4. (Match, $\delta\mathscr{O}$-(Reducible) Couple, Partner, Partnered Item, Bachelor Item). Let $A \in \mathscr{T}$. Let $\bar{s} \equiv s_1 \cdots s_n$ be a segment occurring in A.

- We say that s_i and s_j *match*, when $1 \leqslant i < j \leqslant n$, s_i is a δ-item, s_j is an $\mathscr{O}$-item and the segment $s_{i+1} \cdots s_{j-1}$ is well-balanced. In this case, $s_i s_j$ is a $\delta\mathscr{O}$-*couple* and if $\mathscr{O} = \lambda$ then $s_i s_j$ is a *reducible couple*.

- When s_i and s_j match, we call both s_i and s_j the *partners* in the $\delta\mathscr{O}$-couple. We also say that s_i and s_j are *partnered* items.

- All non-partnered $\mathscr{O}$- (or δ-)items s_k in A are called *bachelor* $\mathscr{O}$- (resp. δ-)items.

3. THE ORDINARY TYPING RELATION AND ITS PROPERTIES

We now introduce some general notions concerning typing rules, which are the same as the usual ones when we do not allow definitions in the context (as is the case in the λ-cube of [Bar92]). When definitions are present, however, the notions are more general.

DEFINITION 3.1 (Declarations, Pseudocontexts, $\subseteq'$).

1. A declaration d is a λ-item $(A\lambda_x)$. We define $\mathtt{subj}(d) = x$, $\mathtt{pred}(d) = A$ and $\underline{d} = \varnothing$. In classical notation, d is written $x : A$.

2. For a definition $d \equiv (B\delta)\,\bar{s}(A\lambda_x)$ we define $\mathtt{subj}(d) = x$, $\mathtt{pred}(d) = A$, $\underline{d} = \bar{s}$ and $\mathtt{def}(d) = B$.

3. We use $d, d_1, d_2, \ldots$ to range over declarations and definitions.

4. A pseudocontext is a concatenation of declarations and definitions such that if $(A\lambda_x)$ and $(B\lambda_y)$ are two different main items of the pseudocontext, then $x \not\equiv y$. We use $\Gamma, \Delta, \Gamma', \Gamma_1, \Gamma_2, \ldots$ to range over pseudocontexts.

5. For Γ a pseudocontext we define

$s \in' \Gamma$ to mean s is a main item occurring in Γ,

$dom(\Gamma) = \{x \in V \mid (A\lambda_x) \in' \Gamma \text{ for some } A\}$,

(classically, $\{x \in V \mid x\colon A \in \Gamma\}$)

$\Gamma\text{-}\mathtt{dec1} = \{s \mid s \text{ is a bachelor main } \lambda\text{-item of } \Gamma\}$,

(classically, any declaration is bachelor)

$\Gamma\text{-}\mathtt{def} = \{\bar{s} \mid \bar{s} \equiv (A\delta)\,\bar{s}_1(B\lambda_x)$ is a main segment of Γ where $\bar{s}_1$ is well-balanced$\}$,

Note that $dom(\Gamma) = \{\mathtt{subj}(d) \mid d \in \Gamma\text{-}\mathtt{dec1} \cup \Gamma\text{-}\mathtt{def}\}$.

6. Define $\subseteq'$ between pseudocontexts as the least reflexive transitive relation satisfying:

- $\Gamma\Delta \subseteq' \Gamma(C\lambda_x)\Delta$ if no λ-item in Δ matches a δ-item in Γ
- $\Gamma\underline{d}\Delta \subseteq' \Gamma\, d\Delta$ if d is a definition
- $\Gamma\bar{s}(A\lambda_x)\Delta \subseteq' \Gamma(D\delta)\,\bar{s}(A\lambda_x)\Delta$ if $(A\lambda_x)$ is bachelor in $\Gamma\bar{s}(A\lambda_x)\Delta$, $\bar{s}$ is well-balanced

EXAMPLE 3.2. A definition $d \equiv (B\delta)(A\lambda_x)$ can be written in classical notation $(\lambda_{x:A}.-)B$ and defines x of type A to be B in $-$. If $d \equiv (B\delta)(C\delta)(D\lambda_y)(A\lambda_x)$ then this is $((\lambda_{y:D}.\lambda_{x:A}.-)C)B$. It is hard to describe in words what this definition means since it is more or less parallel; it has the same overall effect as $(C\delta)(D\lambda_y)(B\delta)(A\lambda_x)$ or classically $(\lambda_{y:D}.(\lambda_{x:A}.-)B)C$.

If $\Gamma \equiv (a\lambda_x)(b\lambda_y)(c\delta)(d\lambda_z)(e\lambda_u)(f\delta)(g\delta)(i\lambda_v)(j\lambda_w)$ then $\Gamma\text{-}\mathtt{dec1} = \{(a\lambda_x), (b\lambda_y), (e\lambda_u)\}$ and $\Gamma\text{-}\mathtt{def} = \{(c\delta)(d\lambda_z), (f\delta)(g\delta)(i\lambda_v)(j\lambda_w), (g\delta)(i\lambda_v)\}$.

Furthermore $\Gamma \subseteq' (*\lambda_r)(a\lambda_x)(b\lambda_y)(h\delta)(c\delta)(d\lambda_z)(k\lambda_{r'})(l\delta)(e\lambda_u)(f\delta)(g\delta)(i\lambda_v)(j\lambda_w)$. Note that $\Gamma \subseteq' \Gamma' \not\Rightarrow \Gamma\text{-}\mathtt{dec1} \subseteq \Gamma'\text{-}\mathtt{dec1}$, but $\Gamma \subseteq' \Gamma' \Rightarrow \Gamma\text{-}\mathtt{def} \subseteq \Gamma'\text{-}\mathtt{def}$.

DEFINITION 3.3 (Statements, Judgments, $\prec$).

1. A statement is of the form $A : B$, A and B are called the subject and the predicate of the statement respectively.

2. When Γ is a pseudocontext and $A : B$ is a statement, we call $\Gamma \vdash A : B$ a judgement, and write $\Gamma \vdash A : B : C$ to mean $\Gamma \vdash A : B \wedge \Gamma \vdash B : C$.

3. For Γ a pseudocontext and $d \in \Gamma\text{-}\mathtt{def} \cup \Gamma\text{-}\mathtt{dec1}$, Γ invites d, notation $\Gamma \prec d$, iff

- Γd is a pseudocontext
- $\Gamma\underline{d} \vdash \mathtt{pred}(d) : S$ for some sort S and $\mathtt{subj}(d) \in V^S$.
- if d is a definition then $\Gamma\underline{d} \vdash \mathtt{def}(d) : \mathtt{pred}(d)$ and $FV(\mathtt{def}(d)) \subseteq dom(\Gamma)$.

$\Gamma \prec d$ holds if d is a "good" declaration *or* a "good" definition with respect to Γ. (Note that $\underline{d}$ is empty if d is a declaration.) Moreover, usually, one requires that $\Gamma(A\lambda_x)$ be a pseudo-context and that $\Gamma \vdash A : S$ before one can use $\Gamma(A\lambda_x)$ in the start and weakening rules (below). With definitions, however, one also has to check that $\Gamma\underline{d} \vdash \mathtt{def}(d) : \mathtt{pred}(d)$ and that $FV(\mathtt{def}(d)) \subseteq dom(\Gamma)$, for obvious reasons, before Γd can be used.

DEFINITION 3.4. (Definitional β-Equality). For all legal contexts Γ we define the binary relation $\Gamma \vdash \cdot =_{\mathtt{def}} \cdot$ to be the equivalence relation generated by

- if $A =_\beta B$ then $\Gamma \vdash A =_{\mathtt{def}} B$
- if $d \in \Gamma\text{-}\mathtt{def}$ and $A, B \in \mathscr{T}$ such that B arises from A by substituting one particular occurrence of $\mathtt{subj}(d)$ in A by $\mathtt{def}(d)$, then $\Gamma \vdash A =_{\mathtt{def}} B$.

Remark 3.5. If no definitions are present in Γ then $\Gamma \vdash A =_{\mathtt{def}} B$ is the same as $A =_\beta B$.

DEFINITION 3.6. Let Γ be a pseudocontext and A be a pseudo-expression.

1. Let $d, d_1, \ldots, d_n$ be declarations and definitions. We define $\Gamma \vdash d$ and $\Gamma \vdash d_1 \cdots d_n$ simultaneously as follows:

- If d is a declaration: $\Gamma \vdash d$ iff

$$\Gamma \vdash \mathtt{subj}(d) : \mathtt{pred}(d).$$

- If d is a definition: $\Gamma \vdash d$ iff

$$\Gamma \vdash \mathtt{subj}(d) : \mathtt{pred}(d) \wedge \Gamma \vdash \mathtt{def}(d) : \mathtt{pred}(d) \wedge$$
$$\Gamma \vdash \underline{d} \wedge \Gamma \vdash \mathtt{subj}(d) =_{\mathtt{def}} \mathtt{def}(d).$$

- $\Gamma \vdash d_1 \cdots d_n$ iff $\Gamma \vdash d_i$ for all $1 \leqslant i \leqslant n$.

2. Γ is called legal if $\exists P, Q \in \mathscr{T}$ such that $\Gamma \vdash P : Q$.

3. $A \in \mathscr{T}$ is called a Γ-term if $\exists B \in \mathscr{T}[\Gamma \vdash A : B$ or $\Gamma \vdash B : A]$. We take $\Gamma\text{-terms} = \{A \in \mathscr{T} \mid \exists B \in \mathscr{T}[\Gamma \vdash A : B \vee \Gamma \vdash B : A]\}$.

4. We take $\Gamma\text{-kinds} = \{A \mid \Gamma \vdash A : \Box\}$ and

$$\Gamma\text{-types} = \{A \in \mathscr{T} \mid \Gamma \vdash A : *\}.$$

5. $A \in \mathscr{T}$ is called a Γ-element if $\exists B \in \mathscr{T}\, \exists S \in \mathscr{S}[\Gamma \vdash A : B$ and $\Gamma \vdash B : S]$. We have two categories of elements: constructors and objects. We take

$\Gamma\text{-constructors} = \{A \in \mathscr{T} \mid \exists B \in \mathscr{T}[\Gamma \vdash A : B : \Box]\}$ and

$\Gamma\text{-objects} = \{A \in \mathscr{T} \mid \exists B \in \mathscr{T}[\Gamma \vdash A : B : *]\}$.

6. $A \in \mathscr{T}$ is called legal if $\exists \Gamma[A \in \Gamma\text{-terms}]$. Moreover, A is an X, if $\exists \Gamma[A \in \Gamma\text{-}Xs]$ for $X \in \{$type, term, kind, object, constructor$\}$.

Note that in 1 we do not have $\Gamma \underline{d} \vdash \mathtt{def}(d) : \mathtt{pred}(d)$ (as is the case for $\Gamma \prec d$) since $\Gamma \vdash d$ intuitively means all information in d is already present in Γ, so extension of Γ with $\underline{d}$ is unnecessary.

In the Cube of [Bar92] and revisited below, the only declarations allowed are of the form $(A\lambda_x)$. Hence there are no definitions. Therefore, $\Gamma \prec d$ is of the form $\Gamma \prec (A\lambda_x)$ and means that $\Gamma \vdash A : S$ for some S and that x is fresh in Γ, A. Moreover, for any $d \equiv (A\lambda_x)$, remember that $\underline{d} \equiv \varnothing$, $\mathtt{subj}(d) \equiv x$ and $\mathtt{pred}(d) \equiv A$. *Hence, in this section, d is a meta-variable for declarations only and* $=_{\mathtt{def}}$ *is the same as* $=_\beta$ *(which is independent of* $\vdash$*).*

3.1. *The Typing Relation*

DEFINITION 3.7 (Axioms and Rules of the Cube: d is a Declaration, $=_{\mathtt{def}}$ is $=_\beta$).

$$\text{(axiom)} \qquad \langle\,\rangle \vdash * : \Box$$

$$\text{(start rule)} \qquad \frac{\Gamma \prec d}{\Gamma d \vdash \mathtt{subj}(d) : \mathtt{pred}(d)} \quad \left(\text{usual notation } \frac{\Gamma \vdash A : S}{\Gamma, x : A \vdash x : A}\, x \text{ fresh}\right)$$

$$\text{(weakening rule)} \qquad \frac{\Gamma \prec d \qquad \Gamma \underline{d} \vdash D : E}{\Gamma d \vdash D : E} \quad \left(\text{or } \frac{\Gamma \vdash A : S \qquad \Gamma \vdash D : E}{\Gamma, x : A \vdash D : E}\, x \text{ fresh}\right)$$

$$\text{(application rule)} \qquad \frac{\Gamma \vdash F : (A\Pi_x) B \qquad \Gamma \vdash a : A}{\Gamma \vdash (a\delta)\, F : B[x := a]}$$

$$\text{(abstraction rule)} \qquad \frac{\Gamma(A\lambda_x) \vdash b : B \qquad \Gamma \vdash (A\Pi_x)\, B : S}{\Gamma \vdash (A\lambda_x)\, b : (A\Pi_x) B}$$

$$\text{(conversion rule)} \qquad \frac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : S \qquad \Gamma \vdash B =_{\mathtt{def}} B'}{\Gamma \vdash A : B'}$$

$$\text{(formation rule)} \qquad \frac{\Gamma \vdash A : S_1 \qquad \Gamma(A\lambda_x) \vdash B : S_2}{\Gamma \vdash (A\Pi_x)\, B : S_2} \quad \text{if } (S_1, S_2) \text{ is a rule}$$

Note that we prefer $\Gamma \prec d$ over $\Gamma \vdash d$. The reason is that we prefer to have only *one* start and weakening rule for both declarations and definitions. The notion $\prec$ as defined in Definition 3.3 takes care of both cases. (Cf. Section 5.1.)

Each of the systems of the Cube is obtained by taking the (S_1, S_2) rules allowed from a subset of $\{(*, *), (*, \Box), (\Box, *), (\Box, \Box)\}$. The basic system is the one where $(*, *)$ is the only possible choice. All other systems have $(*, *)$ plus some combination of $(*, \Box)$, $(\Box, *)$ and $(\Box, \Box)$ for (S_1, S_2). Here is the table which presents the eight systems of the Cube:

System	Set of specific rules			
$\lambda_{\to}$	$(*, *)$			
$\lambda 2$	$(*, *)$	$(\Box, *)$		
λP	$(*, *)$		$(*, \Box)$	
$\lambda P2$	$(*, *)$	$(\Box, *)$	$(*, \Box)$	
$\lambda\underline{\omega}$	$(*, *)$			$(\Box, \Box)$
$\lambda\omega$	$(*, *)$	$(\Box, *)$		$(\Box, \Box)$
$\lambda P\underline{\omega}$	$(*, *)$		$(*, \Box)$	$(\Box, \Box)$
$\lambda P\omega = \lambda C$	$(*, *)$	$(\Box, *)$	$(*, \Box)$	$(\Box, \Box)$

EXAMPLE 3.8. 1. $\vdash_{\lambda 2} (*\Pi_\alpha)(\alpha\Pi_y)\alpha : *$ as we have the rule $(\Box, *)$, but $\not\vdash_{\mathscr{L}} (*\Pi_\alpha)(\alpha\Pi_y)\alpha : \tau$ for any τ where $\mathscr{L} \in \{\lambda_{\to}, \lambda\underline{\omega}, \lambda P, \lambda P\underline{\omega}\}$. (In classical notation, $\vdash_{\lambda 2} \Pi_{\alpha : *}.\Pi_{y : \alpha}.\alpha : *$.)

2. We discuss the following example as a preparation for Example 4.11. $(*\lambda_\sigma)(\sigma\lambda_t)((\sigma\Pi_q) * \lambda_Q)((t\delta)\, Q\lambda_N) \vdash_{\lambda P} (N\delta)(t\delta)(\sigma\lambda_x)((x\delta)\, Q\lambda_y)(y\delta)((x\delta)\, Q\lambda_Z)\, Z : (t\delta)\, Q$, but this derivation cannot be obtained in $\lambda_{\to}$, $\lambda\omega$, $\lambda\underline{\omega}$ or $\lambda 2$ as we need the $(*, \Box)$ rule in order to derive that $(\sigma\Pi_q) * : \Box$ and hence that $((\sigma\Pi_q) * \lambda_Q)$ is allowed in the context. (In classical notation, $\sigma : *, t : \sigma, Q : \Pi_{q : \sigma}.*, N : Qt \vdash_{\lambda P} ((\lambda_{x : \sigma}.\lambda_{y : Qx}.(\lambda_{Z : Qx}.Z)y)t)N : Qt$.)

3. If $\mathscr{L} \in \{\lambda_{\to}, \lambda\underline{\omega}\}$, then $(*\lambda_\beta)(\beta\lambda_{y'}) \not\vdash_{\mathscr{L}} (y'\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_y)(y\delta)(\alpha\lambda_x)x : \beta$ because the term of Part 1 of this example is not typable in $\mathscr{L}$ (note that with definitions, the last nine steps below are replaced by a single one in Example 5.2). Here is how this judgement is derivable in $\lambda 2$. (In classical notation, $\beta : *, y' : \beta \vdash_{\lambda 2} ((\lambda_{\alpha : *}.\lambda_{y : \alpha}.(\lambda_{x : \alpha}.x)y)\beta)y' : \beta$.)

$\vdash * : \Box$ (axiom)

$(* \lambda_\beta) \vdash_{\lambda 2} \beta : * : \Box$ (start resp. weakening rule)

$(* \lambda_\beta)(\beta\lambda_{y'}) \vdash_{\lambda 2} y' : \beta : * : \Box$ (start resp. weakening rule)

$(* \lambda_\beta)(\beta\lambda_{y'})(* \lambda_\alpha) \vdash_{\lambda 2} \alpha : *$ (start)

$(* \lambda_\beta)(\beta\lambda_{y'})(* \lambda_\alpha)(\alpha\lambda_y) \vdash_{\lambda 2} y : \alpha : *$ (start resp. weakening rule)

$(* \lambda_\beta)(\beta\lambda_{y'})(* \lambda_\alpha)(\alpha\lambda_y)(\alpha\lambda_x) \vdash_{\lambda 2} x : \alpha : *$ (start resp. weakening rule)

$(* \lambda_\beta)(\beta\lambda_{y'})(* \lambda_\alpha)(\alpha\lambda_y) \vdash_{\lambda 2} (\alpha\Pi_x)\, \alpha : *$ (formation rule $(*, *)$)

$(* \lambda_\beta)(\beta\lambda_{y'})(* \lambda_\alpha)(\alpha\lambda_y) \vdash_{\lambda 2} (\alpha\lambda_x)\, x : (\alpha\Pi_x)\, \alpha : *$ (abstraction rule)

$(* \lambda_\beta)(\beta\lambda_{y'})(* \lambda_\alpha)(\alpha\lambda_y) \vdash_{\lambda 2} (y\delta)(\alpha\lambda_x) x : \alpha$ (application rule)

$(* \lambda_\beta)(\beta\lambda_{y'})(* \lambda_\alpha) \vdash_{\lambda 2} (\alpha\Pi_y)\, \alpha : *$ (formation rule $(*, *)$)

$(* \lambda_\beta)(\beta\lambda_{y'})(* \lambda_\alpha) \vdash_{\lambda 2} (\alpha\lambda_y)(y\delta)(\alpha\lambda_x)\, x : (\alpha\Pi_y)\, \alpha : *$ (abstraction rule)

$(* \lambda_\beta)(\beta\lambda_{y'}) \vdash_{\lambda 2} (* \Pi_\alpha)(\alpha\Pi_y)\, \alpha : *$ (formation rule $(\Box, *)$)

$(* \lambda_\beta)(\beta\lambda_{y'}) \vdash_{\lambda 2} (* \lambda_\alpha)(\alpha\lambda_y)(y\delta)(\alpha\lambda_x)\, x : (* \Pi_\alpha)(\alpha\Pi_y)\, \alpha$ (abstraction rule)

$(* \lambda_\beta)(\beta\lambda_{y'}) \vdash_{\lambda 2} (\beta\delta)(* \lambda_\alpha)(\alpha\lambda_y)(y\delta)(\alpha\lambda_x)\, x : (\beta\Pi_y)\beta$ (application rule)

$(* \lambda_\beta)(\beta\lambda_{y'}) \vdash_{\lambda 2} (y'\delta)(\beta\delta)(* \lambda_\alpha)(\alpha\lambda_y)(y\delta)(\alpha\lambda_x) x : \beta$ (application rule)

3.2. *Properties of the Ordinary Typing Relation*

Here we list the most important properties of the Cube (see [Bar92]). In the subsequent sections, these properties will be established for the Cube extended with generalised reduction and definition mechanisms.

THEOREM 3.9 (The Church–Rosser Theorem for $\to_\beta$). *If $A \twoheadrightarrow_\beta B$ and $A \twoheadrightarrow_\beta C$ (or if $B =_\beta C$), then for some D, $B \twoheadrightarrow_\beta D$ and $C \twoheadrightarrow_\beta D$*

LEMMA 3.10 (Substitution Lemma for $\vdash$). *Assume $\Gamma(A\lambda_x)\, \Delta \vdash B : C$ and $\Gamma \vdash D : A$ then $\Gamma(\Delta[x := D]) \vdash B[x := D] : C[x := D]$.*

LEMMA 3.11 (Generation Lemma for $\vdash$).

1. $\Gamma \vdash x : C \Rightarrow \exists S_1, S_2 \in \mathscr{S}\ \exists B =_\beta C[\Gamma \vdash B : S_1 \wedge (B\lambda_x) \in {}'\Gamma \wedge \Gamma \vdash C : S_2]$.
2. $\Gamma \vdash (A\Pi_x)\, B : C \Rightarrow \exists S_1,\ S_2 \in \mathscr{S}[\Gamma \vdash A : S_1 \wedge \Gamma(A\lambda_x) \vdash B : S_2 \wedge (S_1, S_2)$ *is a rule* $\wedge\ C =_\beta S_2 \wedge [C \not\equiv S_2 \Rightarrow \exists S[\Gamma \vdash C : S]]]$
3. $\Gamma \vdash (A\lambda_x)\, b : C \Rightarrow \exists S, B\, [\Gamma \vdash (A\Pi_x)\, B : S \wedge \Gamma(A\lambda_x) \vdash b : B \wedge C =_\beta (A\Pi_x)\, B \wedge C \not\equiv (A\Pi_x)\, B \Rightarrow \exists S \in \mathscr{S}[\Gamma \vdash C : S]]$.
4. $\Gamma \vdash (a\delta)\, F : C \Rightarrow \exists A, B, x[\Gamma \vdash F : (A\Pi_x)\, B \wedge \Gamma \vdash a : A \wedge C =_\beta B[x := a] \wedge (B[x := a] \not\equiv C \Rightarrow \exists S \in \mathscr{S}[\Gamma \vdash C : S])]$.

COROLLARY 3.12 (Generation Corollary for $\vdash$).

1. $\Gamma \vdash A : B \Rightarrow \exists S[B \equiv S$ *or* $\Gamma \vdash B : S]$
2. $\Gamma \vdash A : (B_1\Pi_x)\, B_2 \Rightarrow \exists S[\Gamma \vdash (B_1\Pi_x)\, B_2 : S]$
3. *If A is a Γ-term, then A is $\Box$, a Γ-kind or a Γ-element.*

COROLLARY 3.13 (Subtyping for $\vdash$). *Any subterm of a legal term is a legal term.*

THEOREM 3.14 (Subject Reduction for $\vdash$ *and* $\to_\beta$). $\Gamma \vdash A : B \wedge A \twoheadrightarrow_\beta A' \Rightarrow \Gamma \vdash A' : B$.

LEMMA 3.15 (Unicity of Types for $\vdash$ and $\to_\beta$).

1. $\Gamma \vdash A : B_1 \wedge \Gamma \vdash A : B_2 \Rightarrow B_1 =_\beta B_2$
2. $\Gamma \vdash A : B \wedge \Gamma \vdash A' : B' \wedge A =_\beta A' \Rightarrow B =_\beta B'$
3. $\Gamma \vdash B : S, B =_\beta B', \Gamma \vdash A' : B' \Rightarrow \Gamma \vdash B' : S$.

THEOREM 3.16 (Strong Normalisation with Respect to $\vdash$ and $\to_\beta$). *For all $\vdash$ legal terms M, M is strongly normalising with respect to $\twoheadrightarrow_\beta$.*

4. GENERALISING REDUCTION IN THE CUBE

In this section we extend the classical notions of redexes and β-reduction of the Cube and show that all the properties of Section 3.2 except SR are preserved.

4.1. *The Generalised Reduction*

We allow *$\delta\lambda$-couples* to have the same "reduction rights" as *$\delta\lambda$-segments* as follows:

DEFINITION 4.1 (General β-reduction $\hookrightarrow_\beta$ for the Cube). General one-step β-reduction $\hookrightarrow_\beta$, is the least compatible relation generated out of

$$(\text{general } \beta) \quad (B\delta)\, \bar{s}(C\lambda_x)\, A \hookrightarrow_\beta \bar{s}(A[x := B])$$

if $\bar{s}$ is well-balanced.

General $\hookrightarrow\!\!\!\!\rightarrow_\beta$ is the reflexive and transitive closure of $\hookrightarrow_\beta$ and $\approx_\beta$ is the least equivalence relation generated by $\hookrightarrow\!\!\!\!\rightarrow_\beta$.

General β-reduction has firstly been introduced by Nederpelt in [Ned73] in order to prove strong normalisation for a typed lambda calculus inspired by de Bruijn's Authomath.

EXAMPLE 4.2. Cf. Example 1.1. As $(c\delta)(P\lambda_f)(m\delta)(Q\lambda_x)$ is a well-balanced segment, then

$$A \equiv (n\delta)(c\delta)(P\lambda_f)(m\delta)(Q\lambda_x)(R\lambda_y)(y\delta)(x\delta)f$$
$$\hookrightarrow_\beta (c\delta)(P\lambda_f)(m\delta)(Q\lambda_x)(n\delta)(x\delta)f.$$

$(n\delta)(R\lambda_y)$ corresponds to a "generalised" redex in classical notation, which appears after two one-step β-reductions, leading to $(\lambda_{y:R}.cmy)n$. $\hookrightarrow_\beta$ reduces $((\lambda_{f:P}.(\lambda_{x:Q}.\lambda_{y:R}.fxy)m)c)n$ to $(\lambda_{f:P}.(\lambda_{x:Q}.fxn)m)c$. This is difficult in classical notation. We believe that item notation enables one to extend reduction smoothly. Moreover, $\hookrightarrow_\beta$ extends $\rightarrow_\beta$.

LEMMA 4.3. *If $A \rightarrow_\beta B$ then $A \hookrightarrow_\beta B$. Moreover, if $A \hookrightarrow_\beta B$ comes from contracting a $\delta\lambda$-segment then $A \rightarrow_\beta B$.*

Proof. Obvious, as a $\delta\lambda$-segment is an ordinary redex. ∎

LEMMA 4.4. *If $A \hookrightarrow_\beta B$ then $A =_\beta B$.*

Proof. It suffices to consider the case $A \equiv \overline{s_1}(C\delta)\,\bar{s}(D\lambda_x)E$ where the contracted redex is based on $(C\delta)(D\lambda_x)$, $B \equiv \overline{s_1}\bar{s}(E[x:=C])$, and $\bar{s}$ is well-balanced (hence $\mathtt{weight}(\bar{s})$ is even). We prove the lemma by induction on $\mathtt{weight}(\bar{s})$. Case $\mathtt{weight}(\bar{s})=0$ then obvious as $\hookrightarrow_\beta$ coincides with $\rightarrow_\beta$ in this case. Assume the property holds when $\mathtt{weight}(\bar{s})=2n$. Take $\bar{s}$ such that $\mathtt{weight}(\bar{s})=2n+2$. Now, $\bar{s} \equiv (C'\delta)\,\overline{s'}(D'\lambda_y)\,\overline{s''}$ where $\overline{s'}$, $\overline{s''}$ are well-balanced. Assume $x \not\equiv y$ (if necessary, use renaming).

- From $\bar{s}(E[x:=C]) \hookrightarrow_\beta \overline{s'}(\overline{s''}(E[x:=C])[y:=C'])$, IH, and compatibility, $B =_\beta \overline{s_1}\,\overline{s'}(\overline{s''}(E[x:=C])[y:=C']) \equiv \overline{s_1}\,\overline{s'}(\overline{s''}[y:=C'])(E[x:=C][y:=C']) \equiv B''$.
- Moreover, $A \equiv \overline{s_1}(C\delta)(C'\delta)\,\overline{s'}(D'\lambda_y)\overline{s''}(D\lambda_x)\,E \hookrightarrow_\beta \overline{s_1}(C\delta)\overline{s'}(\overline{s''}(D\lambda_x)\,E[y:=C']) \equiv^{BC} \overline{s_1}(C\delta)\,\overline{s'}(\overline{s''}[y:=C'])(D[y:=C']\lambda_x)(E[y:=C']) \equiv B'$. So by IH $A =_\beta B'$.
- $B' \hookrightarrow_\beta \overline{s_1}\,\overline{s'}(\overline{s''}[y:=C'])(E[y:=C'][x:=C])$, $x, y \notin FV(C) \cup FV(C')$ (by BC). Hence, by IH and substitution, $B' =_\beta \overline{s_1}\,\overline{s'}(\overline{s''}[y:=C'])(E[x:=C][y:=C']) \equiv B''$.

Therefore, $A =_\beta B'$, $B' =_\beta B''$ and $B =_\beta B''$, hence $A =_\beta B$. ∎

As a result we see that conversion does not change the typing relation of Section 3.1.

COROLLARY 4.5. *If $A \hookrightarrow\!\!\!\!\rightarrow_\beta B$ then $A =_\beta B$. Moreover, $A \approx_\beta B$ iff $A =_\beta B$.*

4.2. *Properties of Ordinary Typing with Generalised Reduction*

Because $=_\beta$ and $\approx_\beta$ are equivalent, the only lemmas/theorems of Section 3.2 affected by our extension of reductions are those which have $\twoheadrightarrow_\beta$ in their heading. These are CR (Theorem 3.9), SR (Theorem 3.14), Uniqueness of Types (Lemma 3.15), and SN (Theorem 3.16). In this subsection, we show that CR and SN hold for the Cube with $\hookrightarrow\!\!\!\!\rightarrow_\beta$ and that SR holds for $\lambda\underline{\omega}$ and $\lambda{\rightarrow}$ but fails for the other six systems. Unicity of types depends on SR and on the fact that $=_\beta$ is the same as the symmetric transitive closure of $\hookrightarrow\!\!\!\!\rightarrow_\beta$. Hence, we ignore it here as once we prove SR, the proof of Unicity of Types will be exactly that of Lemma 3.15.

THEOREM 4.6 (The Church–Rosser Theorem for $\hookrightarrow\!\!\!\!\rightarrow_\beta$). *If $A \hookrightarrow\!\!\!\!\rightarrow_\beta B$ and $A \hookrightarrow\!\!\!\!\rightarrow_\beta C$, then there exists D such that $B \hookrightarrow\!\!\!\!\rightarrow_\beta D$ and $C \hookrightarrow\!\!\!\!\rightarrow_\beta D$.*

Proof. As $A \hookrightarrow\!\!\!\!\rightarrow_\beta B$ and $A \hookrightarrow\!\!\!\!\rightarrow_\beta C$ then by Corollary 4.5, $A =_\beta B$ and $A =_\beta C$. Hence, $B =_\beta C$ and by CR for $\twoheadrightarrow_\beta$, there exists D such that $B \twoheadrightarrow_\beta D$ and $C \twoheadrightarrow_\beta D$. But, $A \twoheadrightarrow_\beta B$ implies $A \hookrightarrow\!\!\!\!\rightarrow_\beta B$. Hence CR holds for $\hookrightarrow\!\!\!\!\rightarrow_\beta$. ∎

THEOREM 4.7 (Strong Normalisation with Respect to $\vdash$ and $\hookrightarrow\!\!\!\!\rightarrow_\beta$). *For all $\vdash$-legal terms M, M is strongly normalising with respect to $\hookrightarrow\!\!\!\!\rightarrow_\beta$.*

Proof. This is a special case of the proof of Theorem 6.19. ∎

In the rest of this section, $\mathscr{L}$ ranges over $\lambda\underline{\omega}$ and $\lambda{\rightarrow}$. The crucial step in the proof of Subject Reduction in $\lambda\underline{\omega}$ and $\lambda{\rightarrow}$ is proved in the following:

LEMMA 4.8 (Shuffle Lemma for $\lambda\underline{\omega}$ and $\lambda{\rightarrow}$). *$\Gamma \vdash_{\mathscr{L}} \bar{s}_1(A\delta)\,\bar{s}_2 B : C \Leftrightarrow \Gamma \vdash_{\mathscr{L}} \bar{s}_1\bar{s}_2(A\delta)\,B : C$ where $\bar{s}_2$ is well-balanced and the binding variables in $\bar{s}_2$ are not free in A.*

Proof. For a detailed proof, the reader is referred to [BKN94y]. Informally, the reason for this lemma to be true for $\lambda{\rightarrow}$ and $\lambda\underline{\omega}$ is that in these systems, for any legal term of the form $(P\Pi_x)Q$, $x \notin FV(Q)$ (this is not true for the other systems of the cube because of the mixing of levels that comes with the rules $(*, \Box)$ and $(\Box, *)$). Therefore none of the variables of $dom(\overline{s_2})$ can occur free in the type of B which means that B must have a type of the form $(C\Pi_x)D$ and hence B can be applied directly to A. ∎

THEOREM 4.9 (Generalised Subject Reduction for $\lambda\underline{\omega}$ and $\lambda{\rightarrow}$ for $\vdash$ and $\hookrightarrow\!\!\!\!\rightarrow_\beta$). *$\Gamma \vdash_{\mathscr{L}} A : B \wedge A \hookrightarrow\!\!\!\!\rightarrow_\beta A' \Rightarrow \Gamma \vdash_{\mathscr{L}} A' : B$.*

Proof. We prove by simultaneous induction on the generation of $\Gamma \vdash_{\mathscr{L}} A: B$ that

(i) $\Gamma \vdash_{\mathscr{L}} A: B \wedge A \hookrightarrow\!\!\!\!\rightarrow_\beta A' \Rightarrow \Gamma \vdash_{\mathscr{L}} A' : B$
(ii) $\Gamma \vdash_{\mathscr{L}} A: B \wedge \Gamma \hookrightarrow\!\!\!\!\rightarrow_\beta \Gamma' \Rightarrow \Gamma' \vdash_{\mathscr{L}} A : B$,

where $\Gamma \hookrightarrow\!\!\!\!\rightarrow_\beta \Gamma'$ means $\Gamma \equiv \Gamma_1(A\lambda_x)\Gamma_2$, $\Gamma' \equiv \Gamma_1(A'\lambda_x)\Gamma_2$, and $A \hookrightarrow\!\!\!\!\rightarrow_\beta A'$ for some Γ_1, Γ_2, A, A', x. The cases in which

the last rule applied is axiom, start, weakening or conversion are easy (for start: use conversion). We treat the three other cases. Formation: $\Gamma \vdash_{\mathscr{L}} (A_1 \Pi_x) B_1 : S_1$ is a direct consequence of $\Gamma \vdash_{\mathscr{L}} A_1 : S_1$ and $\Gamma(A_1 \lambda_x) \vdash_{\mathscr{L}} B_1 : S_1$; thus (i) comes from IH(i) and IH(ii); (ii) comes from IH(ii). Abstraction: similar to formation. Application: $\Gamma \vdash_{\mathscr{L}} (a\delta)\, F : B_1[x := a]$ is a direct consequence of $\Gamma \vdash_{\mathscr{L}} F : (A_1 \Pi_x) B_1$ and $\Gamma \vdash_{\mathscr{L}} a : A_1$. Now (ii) comes from IH(ii). We consider various cases:

- *Subcase* 1. $(a\delta)F \hookrightarrow_\beta (a\delta)F'$ because $F \hookrightarrow_\beta F'$. Then (i) follows from IH(i).
- *Subcase* 2. $(a\delta)\, F \hookrightarrow_\beta (a'\delta)\, F$ because $a \hookrightarrow_\beta a'$. From IH(i) and application, $\Gamma \vdash (a'\delta)\, F : B_1[x := a']$. Also, from Corollary 3.12, for some S_1: $\Gamma \vdash_{\mathscr{L}} (A_1 \Pi_x)\, B_1 : S_1$ and hence by generation: $\Gamma(A\lambda_x) \vdash_{\mathscr{L}} B_1 : S_1$ and thus by substitution $\Gamma \vdash_{\mathscr{L}} B_1[x := a] : S_1$. Now conversion gives $\Gamma \vdash_{\mathscr{L}} (a'\delta) F : B_1[x := a]$ which proves (i).
- *Subcase* 3. $F \equiv \bar{s}(A'\lambda_y)F'$, $\bar{s}$ well-balanced and $(a\delta)\, F \hookrightarrow_\beta \bar{s}F'[y := a]$. Now, by Lemma 4.8, $\Gamma \vdash_{\mathscr{L}} \bar{s}(a\delta)\, (A'\lambda_y) F' : B_1[x := a]$ and $\bar{s}(a\delta)(A'\lambda_y)\, F' \to_\beta \bar{s}F'[y := a]$, so by SR for $\twoheadrightarrow_\beta$, $\Gamma \vdash_{\mathscr{L}} \bar{s}F'[y := a] : B_1[x := a]$ which proves (i). ∎

Generalised Subject Reduction, however, is not valid for the other systems as the following examples show (note that failure of SR in $\lambda 2$ (resp. λP), means its failure in $\lambda P2$, $\lambda\omega$ and λC (resp. $\lambda P2$, $\lambda P\underline{\omega}$ and λC)):

EXAMPLE 4.10 (SR Does Not Hold in $\lambda 2$ Using $\hookrightarrow\!\!\!\!\twoheadrightarrow_\beta$). $(*\lambda_\beta)(\beta\lambda_{y'}) \vdash_{\lambda 2} (\not\vdash_{\mathscr{L}}$ for $\mathscr{L} \in \{\lambda_\to, \lambda\underline{\omega}\})$ $(y'\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_y)(y\delta)(\alpha\lambda_x)x : \beta$ (see Example 3.8). Moreover, $(y'\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_y)(y\delta)(\alpha\lambda_x)\, x \hookrightarrow_\beta (\beta\delta)(*\lambda_\alpha)(y'\delta)(\alpha\lambda_x)x$. Moreover, $(*\lambda_\beta)(\beta\lambda_{y'}) \not\vdash_{\lambda 2} (\beta\delta)(*\lambda_\alpha)(y'\delta)(\alpha\lambda_x)x : \beta$. Further, $(*\lambda_\beta)(\beta\lambda_{y'}) \not\vdash_{\lambda 2} (\beta\delta)(*\lambda_\alpha)(y'\delta)(\alpha\lambda_x)x : \tau$ for any τ. This is because $(\alpha\lambda_x)\, x : (\alpha\Pi_x)\alpha$ and $y' : \beta$, yet α and β are unrelated and hence we fail in firing the application rule to find the type of $(y'\delta)(\alpha\lambda_x)x$. Looking closer however, one finds that $(\beta\delta)(*\lambda_\alpha)$ is defining α to be β, yet no such information can be used to combine $(\alpha\Pi_x)\alpha$ with β. We will redefine the rules of the Cube to take such information into account.

EXAMPLE 4.11 (SR Does Not Hold in λP Using $\hookrightarrow\!\!\!\!\twoheadrightarrow_\beta$). $(*\lambda_\sigma)(\sigma\lambda_t)((\sigma\Pi_q) * \lambda_Q)((t\delta)\, Q\lambda_N) \vdash_{\lambda P} (N\delta)(t\delta)(\sigma\lambda_x)((x\delta)\, Q\lambda_y)(y\delta)((x\delta)\, Q\lambda_Z)\, Z : (t\delta)\, Q$. Note here that this cannot be derived in $\lambda_\to$, $\lambda\omega$, $\lambda 2$ or $\lambda\underline{\omega}$ (see Example 3.8). And $(N\delta)(t\delta)(\sigma\lambda_x)((x\delta)\, Q\lambda_y)(y\delta)((x\delta)\, Q\lambda_Z)Z \hookrightarrow_\beta (t\delta)(\sigma\lambda_x)(N\delta)((x\delta)\, Q\lambda_Z)Z$. Now, $N : (t\delta)\, Q$, $t : \sigma$, $y : (x\delta)\, Q$, $x : \sigma$, $(t\delta)\, Q \neq_\beta (x\delta)\, Q$, hence $(*\lambda_\sigma)(\sigma\lambda_t)((\sigma\Pi_q) * \lambda_Q)((t\delta)\, Q\lambda_N) \not\vdash_{\lambda P} (t\delta)(\sigma\lambda_x)(N\delta)((x\delta)\, Q\lambda_Z)\, Z : \tau$ for any τ. Here again the reason of failure is similar to the above example. At one stage, we need to match $(x\delta)\, Q$ with $(t\delta)\, Q$ but this is not possible even though we do have the definition segment: $(t\delta)(\sigma\lambda_x)$, which defines x to be t. All this calls for the need to use these definitions.

5. EXTENDING THE CUBE WITH DEFINITION MECHANISMS

We extend the derivation rules so that we can use definitions in the context. The rules remain unchanged except for the addition of one rule, the (def rule), and that the use of $\Gamma \vdash B =_{\mathtt{def}} B'$ in the conversion rule really has an effect now, rather than simply postulating $B =_\beta B'$.

5.1. *The Definition Mechanisms and Extended Typing*

DEFINITION 5.1 (Axioms and rules of the Cube extended with definitions: d ranges over declarations and definitions).

(axiom) $\quad \langle\,\rangle \vdash^e * : \Box$

(start rule) $\quad \dfrac{\Gamma \prec d}{\Gamma d \vdash^e \mathtt{subj}(d) : \mathtt{pred}(d)}$

(weakening rule) $\quad \dfrac{\Gamma \prec d \qquad \Gamma\underline{d} \vdash^e D : E}{\Gamma d \vdash^e D : E}$

(application rule) $\quad \dfrac{\Gamma \vdash^e F : (A\Pi_x)B \qquad \Gamma \vdash^e a : A}{\Gamma \vdash^e (a\delta)\, F : B[x := a]}$

(abstraction rule) $\quad \dfrac{\Gamma(A\lambda_x) \vdash^e b : B \qquad \Gamma \vdash^e (A\Pi_x)\, B : S}{\Gamma \vdash^e (A\lambda_x)\, b : (A\Pi_x)\, B}$

(def rule) $\quad \dfrac{\Gamma d \vdash^e C : D}{\Gamma \vdash^e dC : [D]_d}$ if d is a definition

(conversion rule) $\quad \dfrac{\Gamma \vdash^e A : B \qquad \Gamma \vdash^e B' : S \qquad \Gamma \vdash^e B =_{\mathtt{def}} B'}{\Gamma \vdash^e A : B'}$

(formation rule) $\quad \dfrac{\Gamma \vdash^e A : S_1 \qquad \Gamma(A\lambda_x) \vdash^e B : S_2}{\Gamma \vdash^e (A\Pi_x)\, B : S_2}$ if (S_1, S_2) is a rule

In the above rules, start and weakening play a dual role (below, $d \equiv (B\delta)\ \bar{s}(A\lambda_x)$):

$$\text{(start1 rule)}\quad \frac{\Gamma \vdash^e A : S}{\Gamma(A\lambda_x) \vdash^e x : A}\ x \text{ fresh}$$

$$\text{(start2 rule)}\quad \frac{\Gamma\bar{s} \vdash^e A : S \qquad \Gamma\bar{s} \vdash^e B : A}{\Gamma(B\delta)\ \bar{s}(A\lambda_x) \vdash^e x : A}\ \Gamma d \text{ pseudocontext, } FV(B) \subseteq dom(\Gamma)$$

$$\text{(w1 rule)}\quad \frac{\Gamma \vdash^e A : S \qquad \Gamma \vdash^e D : E}{\Gamma(A\lambda_x) \vdash^e D : E}\ x \text{ fresh}$$

$$\text{(w2 rule)}\quad \frac{\Gamma\bar{s} \vdash^e A : S \qquad \Gamma\bar{s} \vdash^e B : A \qquad \Gamma\bar{s} \vdash^e D : E}{\Gamma(B\delta)\ \bar{s}(A\lambda_x) \vdash^e D : E}\ \Gamma d \text{ pseudocontext, } FV(B) \subseteq dom(\Gamma)$$

Moreover, the (def rule) could also be split into two rules:

$$\text{(def1 rule)}\quad \frac{\Gamma d \vdash^e C : S}{\Gamma \vdash^e dC : S}\ \text{if } d \text{ is a definition}$$

$$\text{(def2 rule)}\quad \frac{\Gamma d \vdash^e C : D \qquad \Gamma d \vdash^e D : S}{\Gamma \vdash^e dC : dD}\ \text{if } d \text{ is a definition}$$

We find it more elegant and compact to write one single rule to represent two different rules. Some people might argue that in the case of the (def rule), we have to split it up in the above rule. We do not agree with this. The (def rule) says that if $C : D$ can be deduced from a concatenation of definitions d, then dC will be of type D where all the sub-definitions in d have been unfolded in D. We do not get type dD in order to avoid things like $d\Box$. It is worth pointing out, however, that our (def rule) is equivalent to the above two by Corollary 5.6.

Note that in the abstraction rule, it follows that $(A\lambda_x)$ is bachelor in $\Gamma(A\lambda_x)$. The reason is that we can show that if Γ is legal then Γ contains no bachelor main δ-items. Hence as $\Gamma \vdash^e (A\Pi_x)\ B : S$, Γ has no bachelor δ-items and so $(A\lambda_x)$ cannot be matched in Γ.

By $\Gamma\underline{d} \vdash^e \mathtt{def}(d) : \mathtt{pred}(d)$ in the (start rule) and (weakening rule), abbreviating $\Box$ (as in $(\Box\delta)(A\lambda_x)$) is not allowed. Also by $\Gamma\underline{d} \vdash^e \mathtt{pred}(d) : S$, abbreviating kinds is not allowed. We believe that this last condition can be omitted but it doesn't seem urgent to do so. Note that the (def rule) does global substitution in the predicate of all the occurrences of subjects in d. The reason is that d no longer remains in the context. In the conversion rule however, substitution is local as Γ keeps all its information (see Definition 3.4).

EXAMPLE 5.2. Here is how the term in Example 3.8 and its $\hookrightarrow_\beta$-contractum are typed in $\lambda 2$. (Note how much more quickly we can type terms once we have definitions. Note also that the derivation given in Example 3.8 is also valid here, yet it is more clear and efficient to use the definitional segments $(y\delta)(\alpha\lambda_x)$ and $(y'\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_y)$. The present derivation is even valid in $\lambda_\to$, because we don't need $(*\lambda_\alpha)(\alpha\lambda_y)(y\delta)(\alpha\lambda_x)x$ to have a type due to the (def rule).)

$\vdash^e_{\lambda 2} * : \Box$ (axiom)

$(*\lambda_\beta) \vdash^e_{\lambda 2} \beta : * : \Box$ (start resp. weakening)

$(*\lambda_\beta)(\beta\lambda_{y'}) \vdash^e_{\lambda 2} y' : \beta : * : \Box$ (start resp. weakening)

$(*\lambda_\beta)(\beta\lambda_{y'})(\beta\delta)(*\lambda_\alpha) \vdash^e_{\lambda 2} y' : \beta : * : \Box, \alpha : *$ (start resp. weakening)

$(*\lambda_\beta)(\beta\lambda_{y'})(\beta\delta)(*\lambda_\alpha) \vdash^e_{\lambda 2} \alpha =_{\mathtt{def}} \beta$ (definition of $=_{\mathtt{def}}$)

$(*\lambda_\beta)(\beta\lambda_{y'})(\beta\delta)(*\lambda_\alpha) \vdash^e_{\lambda 2} y' : \alpha : *$ (conversion)

$(*\lambda_\beta)(\beta\lambda_{y'})(y'\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_y) \vdash^e_{\lambda 2} y : \alpha : *$ (start resp. weakening)

$(*\lambda_\beta)(\beta\lambda_{y'})(y'\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_y)(y\delta)(\alpha\lambda_x) \vdash^e_{\lambda 2} x : \alpha$ (start resp. weakening)

$[\alpha]_{(y'\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_y)(y\delta)(\alpha\lambda_x)} \equiv \alpha[x := y][y := y'][\alpha := \beta] \equiv \beta$

$(*\lambda_\beta)(\beta\lambda_{y'}) \vdash^e_{\lambda 2} (y'\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_y)(y\delta)(\alpha\lambda_x)\ x : \beta$ (def rule)

Also $(*\lambda_\beta)(\beta\lambda_{y'}) \vdash^e_{\lambda 2} (\beta\delta)(*\lambda_\alpha)(y'\delta)(\alpha\lambda_x)\ x : \beta$ as follows (needed derivation steps, including $(*\lambda_\beta)(\beta\lambda_{y'})(\beta\delta)(*\lambda_\alpha) \vdash^e_{\lambda 2} y' : \alpha$ by (*conversion*), are left to the reader):

$(*\lambda_\beta)(\beta\lambda_{y'})(\beta\delta)(*\lambda_\alpha)(y'\delta)(\alpha\lambda_x) \vdash^e_{\lambda 2} x : \alpha$ so by (def rule):

$(*\lambda_\beta)(\beta\lambda_{y'}) \vdash^e_{\lambda 2} (\beta\delta)(*\lambda_\alpha)(y'\delta)(\alpha\lambda_x)x : \alpha[x := y'][\alpha := \beta] \equiv \beta$

EXAMPLE 5.3. Also the term of Example 4.11 can be easily and quickly typed in λP (note that this term cannot be typed in $\lambda\!\rightarrow$ as the term Q cannot):

$$(*\,\lambda_\sigma)(\sigma\lambda_t)((\sigma\Pi_q)*\lambda_Q)((t\delta)\ Q\lambda_N)(N\delta)(t\delta)(\sigma\lambda_x)$$
$$((x\delta)\,Q\lambda_y)(y\delta)((x\delta)\ Q\lambda_Z)\vdash^e_{\lambda P} Z:(x\delta)\,Q$$
$$(*\,\lambda_\sigma)(\sigma\lambda_t)((\sigma\Pi_q)*\lambda_Q)((t\delta)\ Q\lambda_N)$$
$$\vdash^e_{\lambda P}(N\delta)(t\delta)(\sigma\lambda_x)((x\delta)\ Q\lambda_y)(y\delta)((x\delta)\ Q\lambda_Z)\ Z:(t\delta)\,Q$$

Its $\hookrightarrow_\beta$-contractum gets the same type in the same way:

$$(*\,\lambda_\sigma)(\sigma\lambda_t)((\sigma\Pi_q)*\lambda_Q)((t\delta)\ Q\lambda_N)(t\delta)(\sigma\lambda_x)(N\delta)$$
$$((x\delta)\ Q\lambda_Z)\vdash^e_{\lambda P} Z:(x\delta)\,Q$$
$$(*\,\lambda_\sigma)(\sigma\lambda_t)((\sigma\Pi_q)*\lambda_Q)((t\delta)\ Q\lambda_N)$$
$$\vdash^e_{\lambda P}(t\delta)(\sigma\lambda_x)(N\delta)((x\delta)\ Q\lambda_Z)\ Z:(t\delta)\,Q$$

Remark 5.4. We need $\Gamma\vdash^e A=_{\mathtt{def}} B$ instead of $A=_\beta B$ in the conversion rule because we want from $(*\,\lambda_A)(A\delta)(*\,\lambda_x)\vdash^e A:*$ and y is fresh to derive not only $(*\,\lambda_A)(A\delta)(*\,\lambda_x)(A\lambda_y)\vdash^e y:A$ but also $(*\,\lambda_A)(A\delta)(*\,\lambda_x)(A\lambda_y)\vdash^e y:x$. This is not possible if conversion is left with $B=_\beta B'$: how can we ever derive $(*\,\lambda_A)(A\delta)(*\,\lambda_x)(A\lambda_y)\vdash^e y:x$ as $x\neq_\beta A$? If we change to the conversion rule using $=_{\mathtt{def}}$, then we are fine:

$$(*\,\lambda_A)(A\delta)(*\,\lambda_x)(A\lambda_y)\vdash^e y:A$$
$$(*\,\lambda_A)(A\delta)(*\,\lambda_x)(A\lambda_y)\vdash^e x:*$$
$$(*\,\lambda_A)(A\delta)(*\,\lambda_x)(A\lambda_y)\vdash^e x=_{\mathtt{def}} A$$

and so with conversion

$$(*\,\lambda_A)(A\delta)(*\,\lambda_x)(A\lambda_y)\vdash^e y:x$$

From the point of view of efficiency, it may seem unsatisfactory that in the (def rule) definitions are being unfolded in D, since this will usually mean a size explosion of the predicate. The unfolding is not necessary for non-topsorts (i.e., for $D\not\equiv\Box$), however:

LEMMA 5.5. *The following rule is a derived rule*:

(*derived def rule*)

$$\frac{\Gamma d\vdash^e C:D\qquad \Gamma d\vdash^e D:S}{\Gamma\vdash^e dC:dD}\qquad \textit{if } d \textit{ is a definition.}$$

Proof. If $\Gamma d\vdash^e C:D$ then by the (def rule), $\Gamma\vdash^e dC:[D]_d$; if $\Gamma d\vdash^e D:S$ then by the (def rule) $\Gamma\vdash^e dD:S$. Now by conversion $\Gamma\vdash^e dC:dD$ since $\Gamma\vdash^e dD=_{\mathtt{def}}[D]_d$. ∎

COROLLARY 5.6. *The* (*def rule*) *is equivalent to the* (*def*1 *rule*) *and* (*def*2 *rule*) *together.*

Proof. Only the direction from right to left is worth showing. This depends on:

1. correctness of types: if $\Gamma\vdash^e A:B$ then $B\equiv\Box$ or $\Gamma\vdash^e B:S$.
2. if $\Gamma\vdash^e dD:S$ then $\Gamma\vdash^e [D]_d:S$.

Both 1 and 2 are easy to show. Now, if $\Gamma d\vdash^e C:D$ and $D\equiv\Box$ then use the (def2 rule) to get $\Gamma\vdash^e dC:[D]_d$. If $\Gamma d\vdash^e C:D$ and $\Gamma d\vdash^e D:S$ then use the def1 rule, conversion and 2 above to get $\Gamma\vdash^e dC:[D]_d$. ∎

If D is a sort then of course unfolding d in D is not inefficient since d will disappear.

Due to the possibility of using the (def rule) to type a redex, by using the (derived def rule), in some cases it is even possible to circumvent a normally inevitable size explosion: suppose we want to derive in λC a type for the term $(B\delta)(*\,\lambda_\beta)(\beta\lambda_x)((\beta\Pi_y)\beta\lambda_f)(x\delta)f$.

In λC, without definitions, we will have to derive first the type $(*\,\Pi_\beta)(\beta\Pi_x)((\beta\Pi_y)\,\beta\Pi_f)\beta$ for the subterm $(*\,\lambda_\beta)(\beta\lambda_x)((\beta\Pi_x)\,\beta\lambda_f)(x\delta)\,f$, and by the application rule we will finally derive the type $(B\Pi_x)((B\Pi_y)\,B\Pi_f)B$. Note that due to the last applied application rule the term B has been copied four times, which could make the resulting type very large.

Using our type system extended with definitions however, we would first derive the type $(\beta\Pi_x)((\beta\Pi_y)\,\beta\Pi_f)\beta$ for the term $(\beta\lambda_x)((\beta\Pi_y)\,\beta\lambda_f)(x\delta)\,f$, and then by the derived definition rule we would derive the type $(B\delta)(*\,\lambda_\beta)(\beta\Pi_x)((\beta\Pi_y)\,\beta\Pi_f)\beta$ and avoid the substitution of B for β. This is a further evidence for the advantage of using definitions.

5.2. *Properties of the Cube with Definitions*

If we look at Section 3.2 and because we have changed $\vdash$ to $\vdash^e$ but left $\twoheadrightarrow_\beta$ unchanged, we see that all the lemmas and theorems which had $\vdash$ in their heading get affected. In this subsection, we will list these lemmas and theorems for $\vdash^e$ and give their proofs.

LEMMA 5.7 (Free Variable Lemma for $\vdash^e$). *Let Γ be a legal context such that $\Gamma\vdash^e B:C$.*

1. *If d and d' are two different elements of Γ-*`decl` $\cup$ Γ-`def`, *then* `subj`$(d)\not\equiv$`subj`(d').
2. $FV(B)$, $FV(C)\subseteq dom(\Gamma)$.
3. *For s a main item of Γ, $FV(s)\subseteq\{$*`subj`$(d)\mid d\in\Gamma$-`decl` $\cup\,\Gamma$-`def`, *d is to the left of s in $\Gamma\}$*.

Proof. All by induction on the derivation of $\Gamma\vdash^e B:C$. ∎

LEMMA 5.8 (Start Lemma for $\vdash^e$). *Let Γ be a legal context. Then $\Gamma \vdash^e * : \Box$ and $\forall d \in {}'\Gamma[\Gamma \vdash^e d]$.*

Proof. Γ legal $\Rightarrow \exists B, C[\Gamma \vdash^e B : C]$; now use induction on the derivation $\Gamma \vdash^e B : C$. ▮

LEMMA 5.9 (Transitivity Lemma for $\vdash^e$). *Let Γ and Δ be legal contexts. Then: $[\Gamma \vdash^e \Delta \wedge \Delta \vdash^e A : B] \Rightarrow \Gamma \vdash^e A : B$.*

Proof. Induction on the derivation $\Delta \vdash^e A : B$. ▮

LEMMA 5.10 (Definition-Shuffling for $\vdash^e$). *Let d be a definition.*

1. *If $\Gamma d \Delta \vdash^e C =_{\mathtt{def}} D$ then*

$$\Gamma \underline{d}(\mathtt{def}(d)\,\delta)(\mathtt{pred}(d)\,\lambda_{\mathtt{subj}(d)})\,\Delta \vdash^e C =_{\mathtt{def}} D.$$

2. *If $\Gamma d \Delta \vdash^e C : D$ then*

$$\Gamma \underline{d}(\mathtt{def}(d)\delta)(\mathtt{pred}(d)\,\lambda_{\mathtt{subj}(d)})\,\Delta \vdash^e C : D.$$

Proof. 1 is by induction on the generation of $\Gamma(A\delta)\,\bar{s}(B\lambda_x)\,\Delta \vdash^e C =_{\mathtt{def}} D$. 2 is by induction on the derivation of $\Gamma(A\delta)\,\bar{s}(B\lambda_x)\,\Delta \vdash^e C : D$ using 1 for conversion. ▮

LEMMA 5.11 (Thinning for $\vdash^e$).

1. *If $\Gamma_1\Gamma_2 \vdash^e A =_{\mathtt{def}} B$, $\Gamma_1\Delta\Gamma_2$ is a legal context, then $\Gamma_1\Delta\Gamma_2 \vdash^e A =_{\mathtt{def}} B$.*

2. *If Γ and Δ are legal contexts such that $\Gamma \subseteq {}'\Delta$ and if $\Gamma \vdash^e A : B$, then $\Delta \vdash^e A : B$.*

Proof. 1 is by induction on the derivation $\Gamma_1\Gamma_2 \vdash^e A =_{\mathtt{def}} B$. 2 is as follows:

- If $\Gamma\Delta \vdash^e A : B$, $\Gamma \vdash^e C : S$, x is fresh, and no λ-item in Δ is partnered in Γ, then $\Gamma(C\lambda_x)\,\Delta \vdash^e A : B$. Use induction on the derivation $\Gamma\Delta \vdash^e A : B$ and 1 for conversion.
- If $\Gamma\bar{s}\Delta \vdash^e A : B$, $\Gamma\bar{s} \vdash^e C : D : S$, $FV(C) \subseteq dom(\Gamma)$, x is fresh, $\bar{s}$ is well-balanced, then $\Gamma(C\delta)\,\bar{s}(D\lambda_x)\,\Delta \vdash^e A : B$. Use induction on the derivation $\Gamma\bar{s}\Delta \vdash^e A : .B$.
- If $\Gamma\bar{s}(A\lambda_x)\,\Delta \vdash^e B : C$, $(A\lambda_x)$ bachelor, $\bar{s}$ well-balanced, $\Gamma\bar{s} \vdash^e D : A$, $FV(D) \subseteq dom(\Gamma)$, then $\Gamma(D\delta)\,\bar{s}(A\lambda_x)\,\Delta \vdash^e B : C$ is shown by induction on the derivation $\Gamma\bar{s}(A\lambda_x)\,\Delta \vdash^e B : C$ (for conversion, use 1). ▮

LEMMA 5.12 (Substitution Lemma for $\vdash^e$). *Let d be a definition.*

1. *If $\Gamma d\Delta \vdash^e A =_{\mathtt{def}} B$, A and B are $\Gamma d\Delta$-legal terms, then $\Gamma[\Delta]_d \vdash^e [A]_d =_{\mathtt{def}} [B]_d$.*

2. *If B is a Γd-legal term, then $\Gamma d \vdash^e B =_{\mathtt{def}} [B]_d$.*

3. *If $\Gamma(A\delta)(B\lambda_x)\,\Delta \vdash^e C : D$ then $\Gamma\Delta[x := A] \vdash^e C[x := A] : D[x := A]$.*

4. *If $\Gamma(B\lambda_x)\,\Delta \vdash^e C : D$, $\Gamma \vdash^e A : B$, $(B\lambda_x)$ bachelor in Γ, then $\Gamma\Delta[x := A] \vdash^e C[x := A] : D[x := A]$.*

5. *If $\Gamma d\Delta \vdash^e C : D$, then $\Gamma[\Delta]_d \vdash^e [C]_d : [D]_d$.*

Proof. 1. Induction on the derivation rules of $=_{\mathtt{def}}$, 2. Induction on the structure of B, 3. Induction on the derivation rules, using 1, 2, and thinning 4. Idem. 5, use 3. ▮

LEMMA 5.13 (Generation Lemma for $\vdash^e$).

1. *If $\Gamma \vdash^e x : A$ then $\exists B, S, S'$: $(B\lambda_x) \in {}'\Gamma$, $\Gamma \vdash^e B : S$, $\Gamma \vdash^e A =_{\mathtt{def}} B$ and $\Gamma \vdash^e A : S'$.*

2. *If $\Gamma \vdash^e (A\lambda_x)\,B : C$ then for some D and sort S: $\Gamma(A\lambda_x) \vdash^e B : D$, $\Gamma \vdash^e (A\Pi_x)\,D : S$, $\Gamma \vdash^e (A\Pi_x)D =_{\mathtt{def}} C$ and if $(A\Pi_x)\,D \not\equiv C$ then $\Gamma \vdash^e C : S'$ for some sort S'.*

3. *If $\Gamma \vdash^e (A\Pi_x)\,B : C$ then for some sorts S_1, S_2: $\Gamma \vdash^e A : S_1$, $\Gamma \vdash^e B : S_2$, $(S_1, S_2) \in \mathscr{R}$, $\Gamma \vdash^e C =_{\mathtt{def}} S_2$ and if $S_2 \not\equiv C$ then $\Gamma \vdash^e C : S$ for some sort S.*

4. *If $\Gamma \vdash^e (A\delta)\,B : C$, $(A\delta)$ bachelor in B, then for some D, E, $x : \Gamma \vdash^e A : D$, $\Gamma \vdash^e B : (D\Pi_x)\,E$, $\Gamma \vdash^e E[x := A] =_{\mathtt{def}} C$ and if $E[x := A] \not\equiv C$ then $\Gamma \vdash^e C : S$ for some S.*

5. *If $\Gamma \vdash^e \bar{s}A : B$, then $\Gamma\bar{s} \vdash^e A : B$*

Proof. 1, 2, 3, and 4, follow by induction on the derivations (use the thinning lemma). As to 5, we use induction on $\mathtt{weight}(\bar{s})$. Case $\mathtt{weight}(\bar{s}) = 0$: nothing to prove. If we have proven the hypothesis for all segments $\bar{s}$ that obey $\mathtt{weight}(\bar{s}) \leqslant 2n$ and $\mathtt{weight}(\bar{s}) = 2n + 2$, $\bar{s} \equiv \overline{s_1 s_2}$ (neither $\overline{s_1} \equiv \varnothing$ nor $\overline{s_2} \equiv \varnothing$) then by the IH: $\Gamma\overline{s_1} \vdash^e \overline{s_2}A : B$, again applying the induction hypothesis gives $\Gamma\overline{s_1 s_2} \vdash^e A : B$. If we have proven the hypothesis for all segments $\bar{s}$ for which $\mathtt{weight}(\bar{s}) \leqslant 2n$ and $\mathtt{weight}(\bar{s}) = 2n + 2$, $\bar{s} \equiv (D\delta)\,\overline{s_1}(E\lambda_x)$ where $\mathtt{weight}(\overline{s_1}) = 2n$ then an induction on the derivation rules shows that one of the following cases is applicable:

- $\Gamma\bar{s} \vdash^e A : B'$, $\Gamma \vdash^e [B']_{\bar{s}} =_{\mathtt{def}} B$ and if $[B']_{\bar{s}} \not\equiv B$ then $\Gamma \vdash^e B : S$ for some sort S.
- $\Gamma \vdash^e D : F$, $\Gamma \vdash^e \overline{s_1}(E\lambda_x)\,A : (F\Pi_y)G$, $\Gamma \vdash^e B =_{\mathtt{def}} G[y := D]$ and if $G[y := D] \not\equiv B$ then $\Gamma \vdash^e B : S$ for some sort S.

In the first case note that $FV(B) \cap dom(\bar{s}) = \varnothing$ and by thinning $\Gamma\bar{s} \vdash^e [B']_{\bar{s}} =_{\mathtt{def}} B$, by substitution $\Gamma\bar{s} \vdash^e [B']_{\bar{s}} =_{\mathtt{def}} B'$. So $\Gamma\bar{s} \vdash^e B' =_{\mathtt{def}} B$ and by conversion $\Gamma\bar{s} \vdash^e A: B$.

In the second case, by IH, $\Gamma\overline{s_1} \vdash^e (E\lambda_x)\,A : (F\Pi_y)G$. Now 2 tells us that $\Gamma\overline{s_1}(E\lambda_x) \vdash^e A : L$, $\Gamma\overline{s_1} \vdash^e (E\Pi_x)\,L =_{\mathtt{def}} (F\Pi_y)G$ and $(E\Pi_x)L \not\equiv (F\Pi_y)G$ gives $\Gamma\overline{s_1} \vdash^e (F\Pi_y)\,G : S_1$ for some S_1.

This means that $x \equiv y$, $\Gamma\overline{s_1} \vdash^e E =_{\mathtt{def}} F$, $\Gamma\overline{s_1} \vdash^e L =_{\mathtt{def}} G$. Out of $\Gamma\overline{s_1} \vdash^e (E\Pi_x)\,L : S$ we get by 3. that $\Gamma\overline{s_1} \vdash^e E : S_2$ for some sort S_2, thinning gives $\Gamma\overline{s_1} \vdash^e D : F$ so by conversion $\Gamma\overline{s_1} \vdash^e D: E$ and by thinning on $\Gamma\overline{s_1}(E\lambda_x) \vdash^e A : L$ we get $\Gamma\bar{s} \vdash^e A : L$.

Out of $\Gamma \vdash^e B =_{\mathtt{def}} G[x := D]$ we get (thinning and substitution) $\Gamma\bar{s} \vdash^e B =_{\mathtt{def}} G$, out of $\Gamma\overline{s_1} \vdash^e L =_{\mathtt{def}} G$ we get $\Gamma\bar{s} \vdash^e L =_{\mathtt{def}} G$, hence $\Gamma\bar{s} \vdash^e B =_{\mathtt{def}} L$.

Now if $G[y := D] \not\equiv B$ then $\exists S : \Gamma \vdash^e B : S$, and if $G[y := D] \equiv B$ we get out of $\Gamma\overline{s_1} \vdash^e (E\lambda_x)\,A : (F\Pi_y)G$ that $\exists S' : \Gamma\overline{s_1} \vdash^e G : S'$, by thinning and substitution we get

that $\Gamma\bar{s} \vdash^e G[y := D] : S'$. Hence $\exists S : \Gamma\bar{s} \vdash^e B : S$ and by conversion we conlude $\Gamma\bar{s} \vdash^e A : B$. ▮

THEOREM 5.14. (Subject Reduction for $\vdash^e$ and $\to_\beta$). $\Gamma \vdash^e A : B \wedge A \twoheadrightarrow_\beta A' \Rightarrow \Gamma \vdash^e A' : B$.

Proof. We only need to consider $A \to_\beta A'$. Suppose $\Gamma \vdash^e (A\delta)(B\lambda_x)\, C : D$. Then by generation, $\Gamma(A\delta)(B\lambda_x) \vdash^e C : D$, and by substitution we get $\Gamma \vdash^e C[x := A] : D[x := A]$, but as $x \notin FV(D)$, $D[x := A] \equiv D$. The compatibility cases are easy. ▮

This may seem to be too easy a proof for Subject Reduction, but remember that the hard work has already been done in case 5 of the Generation Lemma.

LEMMA 5.15 (Uniqueness of Types for $\vdash^e$).

1. $\Gamma \vdash^e A : B \wedge \Gamma \vdash^e A : B' \Rightarrow \Gamma \vdash^e B =_{\mathtt{def}} B'$.
2. $\Gamma \vdash^e A : B \wedge \Gamma \vdash^e A' : B' \wedge A =_\beta A' \Rightarrow \Gamma \vdash^e B =_{\mathtt{def}} B'$

Proof. 1. By induction on the structure of A using the Generation Lemma.

2. By Church–Rosser and Subject Reduction using 1. ▮

Remark 5.16. We did not prove $\Gamma \vdash^e B : S$, $\Gamma \vdash^e A : B'$, $B =_\beta B' \Rightarrow \Gamma \vdash^e B' : S$. This seems difficult to prove because if $\Gamma \vdash^e B' : S'$ then by Uniqueness of Types $\Gamma \vdash^e S =_{\mathtt{def}} S'$ and it is unclear if $S \equiv S'$. Furthermore, we did not prove $\Gamma \vdash^e A : B$, $\Gamma \vdash^e A' : B'$, $\Gamma \vdash^e A =_{\mathtt{def}} A' \Rightarrow \Gamma \vdash^e B =_{\mathtt{def}} B'$ as here we face similar problems. We claim that one can prove this by showing first that $\Gamma \vdash^e A : B \Rightarrow \Gamma \vdash^e [A]_\Gamma : [B]_\Gamma$, where $[A]_\Gamma$ means all definitions in Γ are to be unfolded in A. We do not need these properties for our theory, however.

THEOREM 5.17 (Strong Normalisation for the Cube with Respect to $\vdash^e$ and $\to_\beta$). *For all $\vdash^e$-legal terms M, M is strongly normalising with respect to $\twoheadrightarrow_\beta$.*

Proof. This is a special case of the proof of Theorem 6.19. ▮

Fact 5.18. Subtyping does not hold for $\vdash^e$. Consider the following derivable judgement:

$$(*\,\lambda_\alpha) \vdash^e (\alpha\delta)(*\,\lambda_\beta)(\beta\lambda_y)(y\delta)(\alpha\lambda_z)\, z : (\alpha\Pi_y)\alpha.$$

The subterm $(*\,\lambda_\beta)(\beta\lambda_y)(y\delta)(\alpha\lambda_z)z$ is not typable: suppose $\Gamma \vdash^e (*\,\lambda_\beta)(\beta\lambda_y)(y\delta)(\alpha\lambda_z)\, z : A$, then by the Generation Lemma, $\Gamma' \vdash^e z : \alpha'$ where $\Gamma' \equiv \Gamma(*\,\lambda_\beta)(\beta\lambda_y)(y\delta)(\alpha\lambda_z)$ and α' satisfies $\Gamma' \vdash^e \alpha =_{\mathtt{def}} \alpha'$ and $\Gamma' \vdash^e \alpha' : S$.

Since Γ cannot contain bachelor δ-items, we know that $(*\,\lambda_\beta)$ is not partnered in Γ', hence $\Gamma' \not\vdash^e \alpha =_{\mathtt{def}} \beta$. But since $(y\delta)(\alpha\lambda_z) \in \Gamma'$-def we know that $\Gamma(*\,\lambda_\beta)(\beta\lambda_y) \vdash^e y : \alpha : S$, also $\Gamma(*\,\lambda_\beta)(\beta\lambda_y) \vdash^e y : \beta$ so by Unicity of Types, $\Gamma(*\,\lambda_\beta)(\beta\lambda_y) \vdash^e \alpha =_{\mathtt{def}} \beta$, contradiction.

The reason for this is that when we typed $(\alpha\delta)(*\,\lambda_\beta)(\beta\lambda_y)(y\delta)(\alpha\lambda_z)z$, we used the context $(*\,\lambda_\alpha)(\alpha\delta)(*\,\lambda_\beta)$ to type $(\beta\lambda_y)(y\delta)(\alpha\lambda_z)z$. This context defines β to be α. Now, to type $(*\,\lambda_\beta)(\beta\lambda_y)(y\delta)(\alpha\lambda_z)z$, the definition $(\alpha\delta)(*\,\lambda_\beta)$ cannot be used. Hence, we do not have all necessary information to type $(*\,\lambda_\beta)(\beta\lambda_y)(y\delta)(\alpha\lambda_z)z$. We do, however, have a partial result:

LEMMA 5.19 (Restricted Subtyping). *If $\Gamma \vdash^e A : B$, A' is a subterm of A such that all bachelor items in A' are also bachelor in A, then A' is legal.*

Proof. We prove by induction on the derivations: if A' is a subterm of Γ or A such that all bachelor items in A' are also bachelor items in Γ respectively A, then A' is legal.

Note that in the case of the (def rule) subterms $\overline{s_2}C$ where $d \equiv \overline{s_1}\,\overline{s_2}$, $\overline{s_1} \not\equiv \varnothing$ do not satisfy the restrictions, since at least one item of $\overline{s_2}$ is bachelor in $\overline{s_2}\,C$ but partnered in dC. ▮

Subterms satisfying the bachelor restriction as in Lemma 5.19 above are more important than those not satisfying it. The reason for this is that the latter terms have an extra abstraction (the newly bachelor λ-item) and hence are Π-types, which makes them more involved, whereas the subterm property is usefull because it tells something about less involved terms.

6. THE CUBE WITH DEFINITIONS AND GENERALISED REDUCTION

Now we extend the type system of Section 5 by changing the reduction $\twoheadrightarrow_\beta$ into $\hookrightarrow\!\!\twoheadrightarrow_\beta$. As was the case in Section 4 the derivation rules stay the same as those with classical β-reduction, hence almost all lemmas that have been proved for the system in Section 5 are still valid, for instance the Generation Lemma and Restricted Subtyping. The only properties that have to be investigated are Church–Rosser, Subject Reduction and Strong Normalisation. We will show now that these properties too are still valid.

THEOREM 6.1 (The Church–Rosser Theorem for $\hookrightarrow\!\!\twoheadrightarrow_\beta$). *If $A \hookrightarrow\!\!\twoheadrightarrow_\beta B$ and $A \hookrightarrow\!\!\twoheadrightarrow_\beta C$, then there exists D such that $B \hookrightarrow\!\!\twoheadrightarrow_\beta D$ and $C \hookrightarrow\!\!\twoheadrightarrow_\beta D$.*

Proof. See Theorem 4.6. ▮

Again the hard work for Subject Reduction is already done in the Generation Lemma:

THEOREM 6.2. (Subject Reduction for $\vdash^e$ and $\hookrightarrow\!\!\twoheadrightarrow_\beta$). *If $\Gamma \vdash^e A : B$ and $A \hookrightarrow\!\!\twoheadrightarrow_\beta A'$ then $\Gamma \vdash^e A' : B$.*

Proof. We only need to consider $A \hookrightarrow_\beta A'$. Suppose $\Gamma \vdash^e dC : D$. Then by generation, $\Gamma d \vdash^e C : D$. Hence by

definition-shuffling (5.10, say $A \equiv \mathtt{def}(d)$, $B \equiv \mathtt{pred}(d)$ and $x \equiv \mathtt{subj}(d)$), $\Gamma \underline{d}(A\delta)(B\lambda_x) \vdash^e C : D$. Hence by substitution $\Gamma \underline{d} \vdash^e C[x := A] : D[x := A]$, and by (def rule) $\Gamma \vdash^e \underline{d}(C[x := A]) : [D[x := A]]_d$, which is $\Gamma \vdash^e d(C[x := A]) : [D]_d$. Now by the variable convention $[D]_d \equiv D$ so we are done. The compatibility cases are easy. ▮

6.1. *Strong Normalisation*

In [BKN94x], we used the technique of [Bar92] to show Strong Normalisation for $\lambda_{\to}$ with extended reduction. Here we shall extend the flexible proof of [Geuvers94]. We do not give the full details, but we only give a rough outline of the adaptations that had to be made to the proof in [Geuvers94]. For details, the reader is referred to [BKN94y]. The proof holds for any $\vdash$ relation of Section 3, for $\vdash^e$ and any reduction relation $\to$ which is CR, contains $\to_\beta$ and is such that the least equivalence relation closed under $\to$ is the same as $=_\beta$.

LEMMA 6.3 (Soundness of $\to$). *If $A, B \in \mathscr{T}$ are legal terms such that $A =_\beta B$ then there is a path of one-step reductions and expansions via legal terms between A and B.*

Proof. By Church–Rosser there exists a term C such that $A \twoheadrightarrow_\beta C$ and $B \twoheadrightarrow_\beta C$. By Subject Reduction for ordinary β-reduction all terms on the path $A \cdots C \cdots B$ are legal. ▮

DEFINITION 6.4. Define the *key redex* of a term M as follows:

1. $(A\delta)(B\lambda_x)C$ has key redex $(A\delta)(B\lambda_x)C$.
2. If M has key redex N, then $(P\delta)M$ has key redex N.

Define $red_{\mathbf{k}}(M)$ to be the term obtained from M by contracting its key redex. Note that not all terms have a key redex and that if a term has a key redex then it is unique.

DEFINITION 6.5. • Define the set of base terms $\mathscr{B}_{\to\!\!\to} \subseteq \Lambda$ by $V \subseteq \mathscr{B}_{\to\!\!\to}$, and if $M \in \mathscr{B}_{\to\!\!\to}$, $N \in SN_{\to\!\!\to}$ then also $(N\delta)M \in \mathscr{B}_{\to\!\!\to}$.

• Call $X \subseteq \Lambda$ *saturated*$_{\to\!\!\to}$ iff: $X \subseteq SN_{\to\!\!\to}$, $\mathscr{B}_{\to\!\!\to} \subseteq X$ and for all $M \in \Lambda$: if $M \in SN_{\to\!\!\to}$ and $red_k(M) \in X$ then also $M \in X$.

• Define $SAT_{\to\!\!\to} = \{X \subseteq \Lambda : X \text{ is saturated}_{\to\!\!\to}\}$

LEMMA 6.6. 1. $SN_{\to\!\!\to} \in SAT_{\to\!\!\to}$.

2. $\forall X \in SAT_{\to\!\!\to} : X \neq \varnothing$.

3. If $N \in SN_{\to\!\!\to}$, $M \in X \in SAT_{\to\!\!\to}$ *and* $x \notin FV(M)$ *then* $(N\delta)(\lambda_x)M \in X$. (*Note here that* [Geuvers94] *takes* $(N\delta)(M\delta)(\lambda_y)(\lambda_x)y$ *instead of* $(N\delta)(\lambda_x)M$. *The first, however, will not fit our purposes, as is explained in Remark* 6.17.)

4. $A, B \in SAT_{\to\!\!\to} \Rightarrow A \to B \in SAT_{\to\!\!\to}$.

5. *If I is a set and $X_i \in SAT_{\to\!\!\to}$ for all $i \in I$, then* $\bigcap_{i \in I} X_i \in SAT_{\to\!\!\to}$.

We define in analogy to [Geuvers94] three maps, first $\mathscr{V}$ of Γ-kinds to the function space of $SAT_{\to\!\!\to}$, then $[\![\ ]\!]_\xi$ of Γ-terms\$\Gamma$-objects to elements of the function space of $SAT_{\to\!\!\to}$, and third $(\!|\ |\!)_\rho$ of Γ-terms to Λ, such that when certain conditions are met we have: $\Gamma \vdash A : B : \Box \Rightarrow [\![A]\!]_\xi \in \mathscr{V}(B)$, $[\![B]\!]_\xi \in SAT_{\to\!\!\to}$ and $\Gamma \vdash A : B \Rightarrow (\!|A|\!)_\rho \in [\![B]\!]_\xi$.

DEFINITION 6.7. Define for all kinds A the set-theoretical interpretation of A as follows:

- $\mathscr{V}(*) = SAT_{\to\!\!\to}$,
- $\mathscr{V}((A\Pi_{x^\Box})B) = \mathscr{V}(A) \to \mathscr{V}(B)$, the function space of $\mathscr{V}(A)$ to $\mathscr{V}(B)$
- $\mathscr{V}((A\Pi_{x^*})B) = \mathscr{V}(B)$
- $\mathscr{V}(dA) = \mathscr{V}(A)$ if d a definition.

Now define $\mathscr{U} = \bigcup \{\mathscr{V}(A) \mid A \text{ is a } \vdash\text{-kind}\}$.

LEMMA 6.8. 1. *If A is a legal constructor, and C is a legal object, then* $\mathscr{V}(A) = \mathscr{V}(A[x^\Box := B])$ *and* $\mathscr{V}(A) = \mathscr{V}(A[x^* := C])$.

2. *If dA is a legal kind, d is a definition, then* $\mathscr{V}([A]_d) = \mathscr{V}(A)$.

DEFINITION 6.9. Let Γ be a $\vdash$-legal context.

- A *Γ-constructor valuation*, notation $\xi \models^\Box \Gamma$, is a map $\xi : V^\Box \to \mathscr{U}$ such that for all $(A\lambda_x) \in' \Gamma$ with A a Γ-kind (i.e., $x \in V^\Box$): $\xi(x) \in \mathscr{V}(A)$.

- If ξ is a constructor valuation, then $[\![\ ]\!]_\xi$: Γ-terms\Γ-objects $\to \mathscr{U}$ is defined inductively,

$$[\![\Box]\!]_\xi := SN_{\to\!\!\to}$$

$$[\![*]\!]_\xi := SN_{\to\!\!\to}$$

$$[\![x^\Box]\!]_\xi := \xi(x^\Box)$$

$$[\![(A\delta)B]\!]_\xi := \begin{cases} [\![B]\!]_\xi\, [\![A]\!]_\xi & \text{if } A \in \Gamma\text{-constructors} \\ [\![B]\!]_\xi & \text{if } A \in \Gamma\text{-objects} \end{cases}$$

$$[\![(A\lambda_x)B]\!]_\xi := \begin{cases} \lambda f \in \mathscr{V}(A).\, [\![B]\!]_{\xi(x:=f)} & \text{if } A \in \Gamma\text{-kinds} \\ [\![B]\!]_\xi & \text{if } A \in \Gamma\text{-types} \end{cases}$$

$$[\![(A\Pi_x)B]\!]_\xi := \begin{cases} [\![A]\!]_\xi \to \bigcap_{f \in \mathscr{V}(A)} [\![B]\!]_{\xi(x:=f)} \\ \quad \text{if } A \in \Gamma\text{-kinds}, x \in V^\Box \\ [\![A]\!]_\xi \to [\![B]\!]_\xi \\ \quad \text{if } A \in \Gamma\text{-types}, x \in V^* \end{cases}$$

where $\xi(x := N)$ is the valuation that assigns $\xi(y)$ to $y \not\equiv x$ and N to x. Furthermore, with $[\![A]\!]_\xi\, [\![B]\!]_\xi$ we mean application of the function $[\![A]\!]_\xi$ onto its argument $[\![B]\!]_\xi$ and by λ we mean function-abstraction.

LEMMA 6.10 (Soundness of $[\![\ ]\!]_\xi$). *If $\Gamma \vdash A : B : \Box$ then for all ξ such that $\xi \models^\Box \Gamma$, we have: $[\![A]\!]_\xi$ and $[\![B]\!]_\xi$ are well defined and $[\![A]\!]_\xi \in \mathscr{V}(B)$, $[\![B]\!]_\xi \in SAT_{\to\!\!\to}$.*

DEFINITION 6.11. If $\xi \models^{\square} \Gamma$, then we call ξ *cute* with respect to Γ if for all $d \in \Gamma$-$\mathtt{def}$ such that $\mathtt{subj}(d) \in V^{\square}$, $\xi(\mathtt{subj}(d)) = [\![\mathtt{def}(d)]\!]_{\xi}$.

LEMMA 6.12. 1. *If $\xi \models^{\square} \Gamma$ and A is Γ-legal, then $[\![A]\!]_{\xi}$ depends only on the values of ξ on the free constructor variables of A.*

2. *If $\xi \models^{\square} \Gamma$ then there is a cute ξ' such that $\xi' \models^{\square} \Gamma$ and $\xi' = \xi$ on the non-definitional constructor variables of* $\mathrm{dom}(\Gamma)$.

3. *If $\xi \models^{\square} \Gamma$ and ξ is cute with respect to Γ then $\Gamma \vdash A =_{\mathtt{def}} B \Rightarrow [\![A]\!]_{\xi} = [\![B]\!]_{\xi}$.*

DEFINITION 6.13. • Let $\xi \models^{\square} \Gamma$ such that ξ is cute with respect to Γ. An object valuation of Γ with respect to ξ, notation $\rho, \xi \models \Gamma$, is a map $\rho: V \to \Lambda$ such that for all $(A\lambda_x) \in' \Gamma$: $\rho(x) \in [\![A]\!]_{\xi}$ (regardless of whether $A \in \Gamma$-kinds or $A \in \Gamma$-types).

• For $\rho, \xi \models \Gamma$ (*note*: this implies that ξ is cute), define $(\!|\ |\!)_{\rho}$: Γ-terms $\to \Lambda$ as follows:

$$
\begin{aligned}
(\!|x|\!)_{\rho} &:= \rho(x)\\
(\!|*|\!)_{\rho} &:= *\\
(\!|\square|\!)_{\rho} &:= \square\\
(\!|(N\delta)M|\!)_{\rho} &:= ((\!|N|\!)_{\rho}\,\delta)(\!|M|\!)_{\rho}\\
(\!|(A\lambda_x)B|\!)_{\rho} &:= ((\!|A|\!)_{\rho}\,\delta)(\lambda_y)(\lambda_x)(\!|B|\!)_{\rho(x:=x)}\\
&\qquad (\text{where } y \notin FV(B))\\
(\!|(A\Pi_x)B|\!)_{\rho} &:= ((\lambda_y)(\!|B|\!)_{\rho(x:=y)}\,\delta)((\!|A|\!)_{\rho}\,\delta)x\\
&\qquad (\text{where } y \notin FV(B))
\end{aligned}
$$

Note that we need BC to ensure that no unwanted bindings occur in the case $(A\Pi_x)B$. The use of x in this case is not essential, we also could reserve one special variable w that should not be used otherwise and define $(\!|(A\Pi_x)B|\!)_{\rho}$ to be $((\lambda_x)(\!|B|\!)_{\rho(x:=x)}\,\delta)((\!|A|\!)_{\rho}\,\delta)w$.

• We define another map $\ulcorner\ \urcorner$: Γ-terms $\to \Lambda$ by

$$
\begin{aligned}
\ulcorner x \urcorner &:= x\\
\ulcorner * \urcorner &:= *\\
\ulcorner \square \urcorner &:= \square\\
\ulcorner (N\delta)M \urcorner &:= (\ulcorner N \urcorner \delta)\ulcorner M \urcorner\\
\ulcorner (A\lambda_x)B \urcorner &:= (\ulcorner A \urcorner \delta)(\lambda_y)(\lambda_x)\ulcorner B \urcorner\\
&\qquad (\text{where } y \notin FV(B))\\
\ulcorner (A\Pi_x)B \urcorner &:= ((\lambda_y)\ulcorner B(x:=y) \urcorner \delta)(\ulcorner A \urcorner \delta)x\\
&\qquad (\text{where } y \notin FV(B))
\end{aligned}
$$

DEFINITION 6.14. Let Γ be a context, $A, B \in \Gamma$-terms. Γ satisfies that A is of type B with respect to $\vdash$ and $\twoheadrightarrow$, notation $\Gamma \models A : B$, iff $\forall \xi, \rho[\ \rho, \xi \models \Gamma \Rightarrow (\!|A|\!)_{\rho} \in [\![B]\!]_{\xi}]$.

LEMMA 6.15. *If $\Gamma(A\delta)\,\underline{d}(B\lambda_x)\Delta$ is a legal context and $\rho, \xi \models \Gamma(A\delta)\,\underline{d}(B\lambda_x)\Delta$ then $(\!|A|\!)_{\rho} \in [\![B]\!]_{\xi}$ and $(\!|B|\!)_{\rho} \in SAT_{\twoheadrightarrow}$.*

2. *$\Gamma d \models A : B \Rightarrow \Gamma \models dA : [B]_d$*

LEMMA 6.16 ($(\!|\ |\!)_{\rho}$ versus $\ulcorner\ \urcorner$).

1. *$\forall M \in \Gamma$-terms, $\forall \rho$: $(\!|M|\!)_{\rho} \equiv \ulcorner M \urcorner[\vec{x} := \rho(\vec{x})]$, where $\vec{x}$ are the free variables of M.*

2. *If $\bar{s}$ is a well-balanced segment then $\ulcorner \bar{s}A \urcorner \equiv \ulcorner \bar{s} \urcorner \ulcorner A \urcorner$ and $\ulcorner \bar{s} \urcorner$ is also well balanced. Moreover, $FV(\ulcorner A \urcorner) = FV(A)$.*

3. *For all $M \in \Gamma$-terms: $\ulcorner M \urcorner$ is strongly normalising $\Rightarrow M$ is strongly normalising.*

Remark 6.17. With this lemma, it becomes clear why we depart from [Geuvers94] by using $\ulcorner (A\lambda_x)B \urcorner$ to be $(\ulcorner A \urcorner \delta)(\lambda_y)(\lambda_x)\ulcorner B \urcorner$ instead of $(\ulcorner A \urcorner \delta)((\lambda_x)\ulcorner B \urcorner \delta)(\lambda_u)(\lambda_v)u$.

Consider, for example, $P \equiv (A\delta)(B\delta)(C\lambda_x)(D\lambda_y)E$ and $Q \equiv (B\delta)(C\lambda_x)E[\ y := A\ ]$. It is obvious that $P \hookrightarrow_{\beta} Q$ and that $\ulcorner P \urcorner \equiv (\ulcorner A \urcorner \delta)(\ulcorner B \urcorner \delta)(\ulcorner C \urcorner \delta)(\lambda_p)(\lambda_x)(\ulcorner D \urcorner)\delta)(\lambda_q)(\lambda_y)\ulcorner E \urcorner \hookrightarrow\!\!\!\!\twoheadrightarrow_{\beta} \ulcorner Q \urcorner \equiv (\ulcorner B \urcorner \delta)(\ulcorner C \urcorner \delta)(\lambda_p)(\lambda_x)\ulcorner E \urcorner[\ y := \ulcorner A \urcorner\ ]$. Yet, if we use the translation of [Geuvers94], then we get $\ulcorner P \urcorner \equiv (\ulcorner A \urcorner \delta)(\ulcorner B \urcorner \delta)(\ulcorner C \urcorner \delta)((\lambda_x)\ulcorner (D\lambda_y)E \urcorner \delta)(\lambda_u)(\lambda_v)u \hookrightarrow\!\!\!\!\twoheadrightarrow_{\beta} \ulcorner Q \urcorner \equiv (\ulcorner B \urcorner \delta)(\ulcorner C \urcorner \delta)((\lambda_x)\ulcorner E \urcorner[\ y := \ulcorner A \urcorner\ ]\delta)(\lambda_s)(\lambda_t)s$.

LEMMA 6.18. $\Gamma \vdash A : B \Rightarrow \Gamma \models A : B$

THEOREM 6.19 (Strong Normalisation for the Cube with Respect to $\vdash^{e}$ and $\hookrightarrow\!\!\!\!\twoheadrightarrow_{\beta}$). *For all $\vdash^{e}$-legal terms M, M is strongly normalising with respect to $\hookrightarrow\!\!\!\!\twoheadrightarrow_{\beta}$.*

Proof. Let M be a $\vdash^{e}$-legal term. Then either $M \equiv \square$ or for some context Γ and term N, $\Gamma \vdash^{e} M : N$. In the first case, clearly M is strongly normalising. In the second case, define canonical elements $c^{A} \in \mathscr{V}(A)$ for all $A \in \Gamma$-kinds as follows:

$$
\begin{aligned}
c^{*} &:= SN_{\hookrightarrow\!\!\!\!\twoheadrightarrow_{\beta}}\\
c^{(A\Pi_x)B} &:= \lambda f \in \mathscr{V}(A).\, c^{B} \qquad \text{if} \quad A \in \Gamma\text{-kinds}, x \in V^{\square}\\
c^{(A\Pi_x)B} &:= c^{B} \qquad \text{if} \quad A \in \Gamma\text{-types}, x \in V^{*}.
\end{aligned}
$$

Take ξ such that $\xi(x) = c^{A}$ whenever $(A\lambda_x) \in' \Gamma$ and $\xi(\mathtt{subj}(d)) = [\![\mathtt{def}(d)]\!]_{\xi}$ whenever $d \in' \Gamma$-$\mathtt{def}$ and take ρ such that $\rho(\mathtt{subj}(d)) = (\!|\mathtt{def}(d)|\!)_{\rho}$ for all subdefinitions d of Γ and $\rho(x) = x$ otherwise. Then $\rho, \xi \models \Gamma$, hence $(\!|M|\!)_{\rho} \in [\![N]\!]_{\xi}$, where $(\!|M|\!)_{\rho} = \ulcorner M \urcorner$ as mentioned in Lemma 6.16. Hence $\ulcorner M \urcorner \in [\![N]\!]_{\xi} \subseteq SN_{\hookrightarrow\!\!\!\!\twoheadrightarrow_{\beta}}$. By Lemma 6.16 now also $M \in SN_{\hookrightarrow\!\!\!\!\twoheadrightarrow_{\beta}}$. ∎

This Theorem also proves SN for the other Cubes in this paper (the Cube extended with nothing, definitions or

$\subset\!\!\rightarrow\!\!\!\!\rightarrow_\beta$) as the legal terms of those Cubes are also legal in the Cube of this section, and SN with respect to $\subset\!\!\rightarrow\!\!\!\!\rightarrow_\beta$ implies SN with respect to $\rightarrow\!\!\!\!\rightarrow_\beta$.

7. COMPARING THE SYSTEM WITH DEFINITIONS TO OTHER SYSTEMS

In this section we compare the type systems generated by $\vdash^e$ with that generated by $\vdash$, and with that of [SP93]. We show a conservativity result which says that in a certain sense, definitions are harmless. That is, even though we can type more terms using $\vdash^e$ than using $\vdash$, whenever a judgment is derivable in a theory $\mathscr{L}$ using definitions and $\vdash^e$, it is also derivable in the theory $\mathscr{L}$ without definitions, using only $\vdash$ and where all the definitions are unfolded. We discuss the effectiveness of derivations and type-checking using definitions. More work has still to be done but it is certain that there is a gain in using definitions.

7.1. *Conservativity*

As we saw in Example 5.2, in the type systems with definitions there are more legal terms. Therefore, it has to be investigated to what extent the set of legal terms has changed. Note first that all derivable judgements in a type system of the λ-cube are derivable in the same type system extended with definitions as we only extended, not changed, the derivation rules. A second remark concerns the bypassing of the *formation rule* by using the *weakening* and *definition rule* instead: In $\lambda 2$ without definitions we can derive the following by using the formation rules $(*, *)$ and $(\Box, *)$ (take $\Gamma \equiv (* \lambda_\beta)(\beta\lambda_y)$):

$$\Gamma \vdash_{\lambda_2} y : \beta : * : \Box$$

$$\Gamma(* \lambda_\alpha) \vdash_{\lambda_2} \alpha : * \qquad \text{(start)}$$

$$\Gamma(* \lambda_\alpha)(\alpha\lambda_x) \vdash_{\lambda_2} x : \alpha : * \qquad \text{(start resp weakening)}$$

$$\Gamma(* \lambda_\alpha) \vdash_{\lambda 2} (\alpha\Pi_x)\ \alpha : * \qquad \text{(formation rule } (*, *))$$

$$\Gamma(* \lambda_\alpha) \vdash_{\lambda 2} (\alpha\lambda_x)\ x : (\alpha\Pi_x)\alpha \qquad \text{(abstraction)}$$

$$\Gamma \vdash_{\lambda 2} (* \Pi_\alpha(\alpha\Pi_x)\ \alpha : * \qquad \text{(formation rule } (\Box, *))$$

$$\Gamma \vdash_{\lambda 2} (* \lambda_\alpha)(\alpha\lambda_x)\ x : (* \Pi_\alpha)(\alpha\Pi_x)\alpha \qquad \text{(abstraction)}$$

$$\Gamma \vdash_{\lambda 2} (\beta\delta)(* \lambda_\alpha)(\alpha\lambda_x)\ x : (\beta\Pi_x)\beta \qquad \text{(application, we already knew } \Gamma \vdash_{\lambda 2} \beta : *)$$

$$\Gamma \vdash_{\lambda 2} (y\delta)(\beta\delta)(* \lambda_\alpha(\alpha\lambda_x)\ x : \beta \qquad \text{(application, we already knew } \Gamma \vdash_{\lambda 2} y : \beta).$$

It is not possible to derive this judgement in $\lambda_{\rightarrow}$ as $(\Box, *)$ is needed. Using the observation that $(y\delta)(\beta\delta)(* \lambda_\alpha)(\alpha\lambda_x)x$ can be seen as x with two definitions added, we can derive the judgement in a system with definitions without having to use the $(*, *)$ and $(\Box, *)$:

$$\Gamma \vdash^e_{\lambda_{\rightarrow}} y : \beta : * : \Box$$

$$\Gamma(\beta\delta)(* \lambda_\alpha) \vdash^e_{\lambda_{\rightarrow}} y : \beta, \alpha : * \qquad \text{(weakening resp. start)}$$

$$\Gamma(\beta\delta)(* \lambda_\alpha) \vdash^e_{\lambda_{\rightarrow}} \alpha =_{\mathtt{def}} \beta \qquad \text{(use the definition in the context)}$$

$$\Gamma(\beta\delta)(* \lambda_\alpha) \vdash^e_{\lambda_{\rightarrow}} y : \alpha \qquad \text{(conversion)}$$

$$\Gamma(y\delta)(\beta\delta)(* \lambda_\alpha)(\alpha\lambda_x) \vdash^e_{\lambda_{\rightarrow}} x : \alpha \qquad \text{(start)}$$

$$\Gamma \vdash^e_{\lambda_{\rightarrow}} (y\delta)(\beta\delta)(* \lambda_\alpha)(\alpha\lambda_x)\ x : \alpha[x := y][\alpha := \beta] \equiv \beta \qquad \text{(definition rule)}.$$

This example shows that in $\lambda_{\rightarrow_{\mathtt{def}}}$ we have more legal judgements than in $\lambda_{\rightarrow}$. Now take the judgement $\Gamma \vdash (\beta\delta)(* \lambda_\alpha)(M\lambda_x)\ x : (M\Pi_x)\,M$ where $M \equiv (y\delta)(\beta\lambda_z)(\beta\delta)(* \lambda_\gamma)\gamma$ and $\Gamma \equiv (* \lambda_\beta)(\beta\lambda_y)$. This can be derived in λC using $(\Box, \Box)$, $(\Box, *)$, $(*, \Box)$ and $(*, *)$ as follows:

$\Gamma \vdash_{\lambda C} \beta : * : \Box$

$\Gamma(* \lambda_\alpha) \vdash_{\lambda C} \beta : * : \Box$ (weakening)

$\Gamma(* \lambda_\alpha)(\beta\lambda_z) \vdash_{\lambda C} z : \beta : * : \Box$ (start resp. weakening)

$\Gamma(* \lambda_\alpha)(\beta\lambda_z)(* \lambda_\gamma) \vdash_{\lambda C} \gamma : * : \Box$ (start resp. weakening)

$\Gamma(* \lambda_\alpha)(\beta\lambda_z) \vdash_{\lambda C} (* \Pi_\gamma) * : \Box$ (formation rule $(\Box, \Box)$)

$\Gamma(* \lambda_\alpha)(\beta\lambda_z) \vdash_{\lambda C} (* \lambda_\gamma)\, \gamma : (*\Pi_\gamma) *$ (abstraction)

$\Gamma(* \lambda_\alpha)(\beta\lambda_z) \vdash_{\lambda C} (\beta\delta)(* \lambda_\gamma)\, \gamma : *$ (application)

$\Gamma(* \lambda_\alpha) \vdash_{\lambda C} (\beta\Pi_z) * : \Box$ (formation rule $(*, \Box)$)

$\Gamma(* \lambda_\alpha) \vdash_{\lambda C} (\beta\lambda_z)(\beta\delta)(* \lambda_\gamma)\, \gamma : (\beta\Pi_z) *$ (abstraction)

$\Gamma(* \lambda_\alpha) \vdash_{\lambda C} M : *$ (application, $M \equiv (y\delta)(\beta\lambda_z)(\beta\delta)(* \lambda_\gamma)\gamma$)

$\Gamma(* \lambda_\alpha)(M\lambda_x) \vdash_{\lambda C} x : M : *$ (start resp. weakening)

$\Gamma(* \lambda_\alpha) \vdash_{\lambda C} (M\Pi_x)\, M : *$ (formation rule $(*, *)$)

$\Gamma(* \lambda_\alpha) \vdash_{\lambda C} (M\lambda_x)\, x : (M\Pi_x) M$ (abstraction)

$\Gamma \vdash_{\lambda C} (* \Pi_\alpha)(M\Pi_x)\, M : *$ (formation rule $(\Box, *)$)

$\Gamma \vdash_{\lambda C} (* \lambda_\alpha)(M\lambda_x)\, x : (* \Pi_\alpha)(M\Pi_x)\, M$ (abstraction)

$\Gamma \vdash_{\lambda C} (\beta\delta)(* \lambda_\alpha)(M\lambda_x)\, x : (M\Pi_x)\, M$ (application).

It is impossible to derive this judgement in any other system of the cube than λC, as all four formation rules are needed. We can, however, derive this judgement in $\lambda_{\to\,\mathtt{def}}$:

$\Gamma \vdash^e_{\lambda_\to} \beta : * : \Box$

$\Gamma(\beta\delta)(* \lambda_\alpha) \vdash^e_{\lambda_\to} \beta : * : \Box$ (weakening)

$\Gamma(\beta\delta)(* \lambda_\alpha)(y\delta)(\beta\lambda_z) \vdash^e_{\lambda_\to} \beta : * : \Box$ (weakening)

$\Gamma(\beta\delta)(* \lambda_\alpha)(y\delta)(\beta\lambda_z)(\beta\delta)(* \lambda_\gamma) \vdash^e_{\lambda_\to} \gamma : *$ (weakening)

$\Gamma(\beta\delta)(* \lambda_\alpha) \vdash^e_{\lambda_\to} (y\delta)(\beta\lambda_z)(\beta\delta)(* \lambda_\gamma)\, \gamma : *[\gamma := \beta][z := y]$ i.e. $M : *$ (definition rule)

$\Gamma(\beta\delta)(* \lambda_\alpha)(M\lambda_x) \vdash^e_{\lambda_\to} x : M : *$ (start, weakening)

$\Gamma(\beta\delta)(* \lambda_\alpha) \vdash^e_{\lambda_\to} (M\Pi_x)\, M : *$ (formation $(*, *)$)

$\Gamma(\beta\delta)(* \lambda_\alpha) \vdash^e_{\lambda_\to} (M\lambda_x)\, x : (M\Pi_x)\, M$ (abstraction)

$\Gamma \vdash^e_{\lambda_\to} (\beta\delta)(* \lambda_\alpha)(M\lambda_x)\, x : (M\Pi_x)\, M[\alpha := \beta] \equiv (M\Pi_x) M$ (definition rule).

This shows that in every system of the cube (except λC), definitions derive more judgements.

As was shown in Example 5.2, $(* \lambda_\beta)(\beta\lambda_{y'}) \vdash^e_{\lambda 2} (\beta\delta)(* \lambda_\alpha)(y'\delta)(\alpha\lambda_x)\, x : \beta$ is derivable in $\lambda_2{}^{\mathtt{def}}$ and hence is also derivable in $\lambda C_{\mathtt{def}}$, but this judgement cannot be derived in λC as y is of type β and not of type α. At first sight this might cause the reader to suspect type systems with definitions of having too many derivable judgements. However, we have a conservativity result stating that a judgement that can be derived in $\mathscr{L}_{\mathtt{def}}$ can be derived in $\mathscr{L}$ when all definitions in the whole judgement have been unfolded.

DEFINITION 7.1. For $\Gamma \vdash^e A : B$ a judgement we define the unfolding of $\Gamma \vdash^e A : B$, $[\Gamma \vdash^e A : B]^u$ to be the judgement obtained from $\Gamma \vdash^e A : B$ in the following way:

- first, mark all visible $\delta\lambda$-couples in Γ, A and B,
- second, contract in Γ, A and B all these marked $\delta\lambda$-couples.

When $\Gamma \equiv \cdots (C\delta)\, \vec{s}(D\lambda_x) \cdots$, contracting $(C\delta)(D\lambda_x)$ amounts to substituting all free occurrences of x in the scope of λ_x by C; these free occurrences may also be in one of the terms A and B. The result is independent of the order in which the redexes are contracted, as one can see this unfolding

as a *complete development* (see [Barendregt84]) in a certain sense.

EXAMPLE 7.2. $[(*\lambda_\beta)(\beta\lambda_y)(y\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_x)(\alpha\lambda_z) \vdash^e ((\alpha\lambda_u)u\delta)((\alpha\Pi_u)\beta\lambda_v)(x\delta)v : \alpha]^u$ is $(*\lambda_\beta)(\beta\lambda_y)((\alpha\lambda_z)[x:=y][\alpha:=\beta]) \vdash^e (((x\delta)v)[v:=(\alpha\lambda_u)u])[x:=y][\alpha:=\beta] : \alpha[x:=y][\alpha:=\beta]$, which is $(*\lambda_\beta)(\beta\lambda_y)(\beta\lambda_z) \vdash^e (y\delta)(\beta\lambda_u)u : \beta$. Note that the resulting context contains only λ-items and that the resulting subject and predicate need not be in normal form.

THEOREM 7.3 (Conservativity of definitions). *Let $\mathscr{L}$ be one of the systems of the Cube, Γ a context (possibly with definitions), and A, B pseudoterms. If $\Gamma \vdash^e_{\mathscr{L}} A : B$ then*:

1. $[\Gamma \vdash^e_{\mathscr{L}} A : B]^u$
2. $\Gamma' \vdash_{\mathscr{L}} A' : B'$, *where* $\Gamma' \vdash^e_{\mathscr{L}} A' : B'$ *is* $[\Gamma \vdash^e_{\mathscr{L}} A : B]^u$.

Proof. 1, 2 are both proven by induction on the derivation of $\Gamma \vdash^e_{\mathscr{L}} A : B$. *Axiom*, *abstraction*, and *formation rules* are easy; we treat the other cases for 2.

- The last rule applied is the *start rule*. Then $\Gamma d \vdash^e_{\mathscr{L}} \mathtt{subj}(d) : \mathtt{pred}(d)$ as a consequence of $\Gamma \prec d$. Now if $d \equiv (A\lambda_x)$ then by IH $\Gamma' \vdash_{\mathscr{L}} A' : S$ (S a sort, x fresh) so by the *start rule* $\Gamma'(A'\lambda_x) \vdash_{\mathscr{L}} x : A'$. On the other hand, if d is a definition, say $d \equiv (A\delta)\,\underline{d}(B\lambda_x)$, then by IH $(\Gamma\underline{d})' \vdash_{\mathscr{L}} A' : B' : S$ (S a sort), which is $\Gamma' \vdash_{\mathscr{L}} A' : B' : S$ as d will be fully unfolded, and the unfolding of $\Gamma d \vdash^e_{\mathscr{L}} \mathtt{subj}(d) : \mathtt{pred}(d)$ is $\Gamma' \vdash_{\mathscr{L}} \mathtt{def}(d)' : \mathtt{pred}(d)'$ which is $\Gamma' \vdash_{\mathscr{L}} A' : B'$, so we are done.

- The last rule applied is the *weakening rule*, say $\Gamma d \vdash^e_{\mathscr{L}} D : E$ as a consequence of $\Gamma \prec d$ and $\Gamma\underline{d} \vdash^e_{\mathscr{L}} D : E$. Because $\mathtt{subj}(d)$ is fresh we have that $(\Gamma d)' \vdash_{\mathscr{L}} D' : E'$ is the same as $(\Gamma\underline{d})' \vdash_{\mathscr{L}} D' : E'$ so by IH we are done.

- The last rule applied is the *application rule*. Then $\Gamma \vdash^e_{\mathscr{L}} (a\delta)\,F : B[x:=a]$ as a consequence of $\Gamma \vdash^e_{\mathscr{L}} F : (A\Pi_x)B$ and $\Gamma \vdash^e_{\mathscr{L}} a : A$. By IH and the *application rule* we get $\Gamma' \vdash_{\mathscr{L}} (a'\delta)\,F' : B'[x:=a']$. Now by subject reduction also $\Gamma' \vdash_{\mathscr{L}} ((a'\delta)\,F')' : B'[x:=a']$. If $B'[x:=a'] \equiv (B'[x:=a'])'$ then we are done, otherwise, by the Generation Corollary $\Gamma' \vdash_{\mathscr{L}} B'[x:=a'] : S$ for some sort S, so by subject reduction $\Gamma' \vdash_{\mathscr{L}} (B'[x:=a'])' : S$ and as $B'[x:=a'] =_\beta (B'[x:=a'])'$ by *conversion* we are done.

- The last rule applied is the *conversion rule*. Then $\Gamma \vdash^e_{\mathscr{L}} A : B_2$ as a consequence of $\Gamma \vdash^e_{\mathscr{L}} A : B_1$, $\Gamma \vdash^e_{\mathscr{L}} B_2 : S$ and $\Gamma \vdash^e_{\mathscr{L}} B_1 =_{\mathtt{def}} B_2$. Now $\Gamma \vdash^e_{\mathscr{L}} B_1 =_{\mathtt{def}} B_2$ implies $B'_1 =_\beta B'_2$ because if C results from D by locally unfolding a definition of Γ then $C' \equiv D'$, so the result follows by IH.

- The last rule applied is the *definition rule*. Then $\Gamma \vdash^e_{\mathscr{L}} dc : [D]_d$ as a consequence of $\Gamma d \vdash C : D$. By IH, $\Gamma' \vdash_{\mathscr{L}} [C']_d : [D']_d$ which is the unfolding of $\Gamma \vdash^e_{\mathscr{L}} dc : [D]_d$.

COROLLARY 7.4. *Let Γ, A, B be definition-free. If $\Gamma \vdash^e_{\mathscr{L}} A : B$ then $\Gamma \vdash_{\mathscr{L}} A : B$.*

Remark 7.5. It is not sufficient in Theorem 7.3 to unfold all the definitions in the context only, because a redex in the subject may have been used to change the type when it was still in the context. This is illustrated by $(*\lambda_\beta)(\beta\lambda_y) \vdash^e_{\lambda\to} (\beta\delta)(*\lambda_\alpha)(y\delta)(\alpha\lambda_x)\,x : \beta$ which cannot be derived using $\vdash_{\lambda\to}$. However, this judgement where all the definitions are unfolded in context, subject and predicate, is derivable using $\vdash^e$. That is, $(*\lambda_\beta)(\beta\lambda_y) \vdash_{\lambda\to} y : \beta$.

7.2. *Shorter Derivations and Type Checking*

As already noted, derivations using the definitions need considerably less steps to derive a judgment that can also be derived without definitions. This is due to the fact that redexes in the term to be derived can be introduced by the *def rule*, which bypasses the formation rule.

Type checking with definitions at first sight seems to be more difficult than in the systems of the λ-cube of Barendregt. Consider for instance the type-checking problem $\Gamma \vdash^e (\sigma\delta)(*\lambda_\alpha)(x\delta)(\alpha\delta)\,P : ?$ where $\Gamma \equiv (*\lambda_\sigma)((*\Pi_\beta)(\beta\Pi_y)\beta\lambda_P)(\sigma\lambda_x)$.

Note that this problem is not solvable in the non-extended systems, since $(\alpha\delta)\,P : \alpha \to \alpha$ and $x : \sigma$, so $(x\delta)(\alpha\delta)P$ is not typable. In the extended systems, the only thing a typechecking algorithm can do is trying to solve $\Gamma(\sigma\delta)(*\lambda_\alpha) \vdash^e (x\delta)(\alpha\delta)\,P : ?$,

which is equivalent to finding A, B, y such that

$$\begin{cases} \Gamma(\sigma\delta)(*\lambda_\alpha) \vdash^e (\alpha\delta)\,P : (A\Pi_y)B \\ \Gamma(\sigma\delta)(*\lambda_\alpha) \vdash^e x : A, \end{cases}$$

which again is equivalent to finding z, C such that

$$\begin{cases} \Gamma(\sigma\delta)(*\lambda_\alpha) \vdash^e P : (C\Pi_z)(A\Pi_y)B \\ \Gamma(\sigma\delta)(*\lambda_\alpha) \vdash^e \alpha : C \\ \Gamma(\sigma\delta)(*\lambda_\alpha) \vdash^e x : A. \end{cases}$$

Now $\Gamma(\sigma\delta)(*\lambda_\alpha) \vdash^e P : (*\Pi_\beta)(\beta\Pi_y)\beta$ and $\Gamma(\sigma\delta)(*\lambda_\alpha) \vdash^e \alpha : *$, hence $\Gamma(\sigma\delta)(*\lambda_\alpha) \vdash^e (\alpha\delta)\,P : (\alpha\Pi_y)\alpha$ and $\Gamma(\sigma\delta)(*\lambda_\alpha) \vdash^e x : \sigma$.

Now we face the problem of converting $(\alpha\Pi_y)\alpha$ to $(\sigma\Pi_y)$? or σ to α in the context $\Gamma(\sigma\delta)(*\lambda_\alpha)$ and this is easily done by unfolding the definition $(\sigma\delta)(*\lambda_\alpha)$ in $(\alpha\Pi_y)\alpha$, giving $\Gamma(\sigma\delta)(*\lambda_\alpha) \vdash^e (\alpha\delta)\,P : (\sigma\Pi_y)\sigma$ and hence $\Gamma \vdash^e (\sigma\delta)(*\lambda_\alpha)(x\delta)(\alpha\delta)\,P : \sigma$.

We saw that typechecking gave rise to locally unfolding a definition in the type $(\alpha\Pi_y)\alpha$. This is something new in comparison with typechecking in the λ-cube of Barendregt where only reduction to (weak head-) normal form of the type is needed. Now if we want to typecheck a redex it appears to be a reasonable strategy to consider it as a definition

since it is not easy to see whether a redex in a term can be typed without the (def rule).

So for example, when typechecking $(t\delta)(\sigma\lambda x)(Q\delta)(x\delta)(\sigma\delta)P$ in our extended system with context Γ such that $\Gamma \vdash^e ((*\,\Pi_\alpha)(\alpha\Pi_y)(\alpha\Pi_z)\,\alpha\lambda_P)$ and $\Gamma \vdash^e (*\,\lambda_\sigma)$, an automated type checker will try to solve

$$\Gamma(t\delta)(\sigma\lambda_x) \vdash^e (Q\delta)(x\delta)(\sigma\delta)\,P : ? \text{ instead of}$$

$$\begin{cases} \Gamma \vdash^e (\sigma\lambda_x)(Q\delta)(x\delta)(\sigma\delta)\,P : (\sigma\Pi_x)\,A \\ \Gamma \vdash^e t : \sigma. \end{cases}$$

As a result, something like

$$\begin{cases} \Gamma(t\delta)(\sigma\lambda_x) \vdash^e (x\delta)(\sigma\delta)\,P : (A\Pi_y)\,B \\ \Gamma(t\delta)(\sigma\lambda_x) \vdash^e Q : A' \end{cases}$$

will be derived and now it has to be checked whether $\Gamma(t\delta)(\sigma\lambda_x) \vdash^e A =_{\mathtt{def}} A'$. In case the original redex was not a definition, $A =_{\mathtt{def}} A'$ can be established without using the context definition $(t\delta)(\sigma\lambda_x)$. Hence we conjecture that an intelligent typecheck algorithm can avoid needless extra work by unfolding definitions only as a last resort. Further research has yet to be done in this direction.

7.3. *Comparison with the Systems of the Barendregt Cube*

Here we discuss the (dis)advantages of our extended typing systems to the cube systems.

In the extended typing systems we can reason with definitions in the context (which is very natural to do): we can add definitions to the context in which we reason (the *start rule* and *weakening rule*), we can eliminate definitions in the context (the *def rule*), and we can unfold a definition in the context locally in the type (the *conversion rule*).

Furthermore, in the terms, there are more visible (and subject to contraction) redexes.

If one considers one of the seven lower systems in the λ-cube, some abstractions are forbidden, for instance in $\lambda P\underline{\omega}$ the abstraction of a term over a type is not allowed (this abstraction corresponds to universal quantification in logic). Intuitively such a quantification need not be forbidden if it is immediately instantiated by an application, as is the case in $(\lambda_{\alpha:*}.(\lambda_{x:\alpha}.x))\beta$. However, in the system $\lambda P\underline{\omega}$ this term is untypable as the subterm $\lambda_{\alpha:*}.(\lambda_{x:\alpha}.x)$ should have type $\Pi_{\alpha:*}.(\Pi_{x:\alpha}.\alpha)$, which is forbidden as $(\Box, *)$ is not allowed.

Now in our extended typing system $\lambda P\underline{\omega}_e$ we can type the term $(\lambda_{\alpha:*}.(\lambda_{x:\alpha}.x))\beta$ by using the *def rule*: from $(*\,\lambda_\beta)(\beta\delta)(*\,\lambda_\alpha) \vdash^e (\alpha\lambda_x)x : (\alpha\Pi_x)x$ we may conclude $(*\,\lambda_\beta) \vdash^e (\beta\delta)(*\,\lambda_\alpha)(\alpha\lambda_x)\,x : (\beta\Pi_x)x$. Note that the use of the formation rule $(\Box, *)$ is avoided.

By this property, the extended type systems are closer to intuition than the systems of the cube as there are more (intuitively correct) derivable inhabitants of certain types.

In the last paragraphs of [Bar92], a term is shown which is an inhabitant of $\bot$ in λU^-. It is remarked that the given term is not legal due to the fact that some definitions are being used that shorten the term by a factor 72. In the system λU^- extended with definitions, this term is still illegal due to the restriction that definitions should have a typable `pred`. This suggests that there is a need for an even more flexible use of definitions, such that also terms on the highest typing levels can be abbreviated.

7.4. *Comparison with the Type Systems of Poll and Severi*

When we compare the extended type systems to those of Poll and Severi (see [SP93]), we observe the following differences.

1. In the systems of [SP93], the definition of pseudoterms has been adapted, not only the usual variables, abstractions and applications are pseudoterms, but definitions, i.e. terms of the form $x = a : A$ `in` B are added. A new reduction relation has to be introduced to be able to unfold these definitions (locally in the predicate of the judgment). This means Church–Rosser has to be shown again.

In our approach, we treat definitions not much different from β-redexes, hence the syntax of pseudoterms remains the same. We only change the syntax of contexts and extend β-equality in a natural way to be able to use the definitions in the context and unfold them locally in the predicate of a judgement. Church–Rosser remains unchanged.

2. [SP93] have a rule that takes a definition out of the context and puts it in front of the term and type, provided that the type is not a topsort. In our extended system however, we only put the definition in front of the term and unfold it in the type. As we already noted (cf. Lemma 5.5), the (derived def rule) allows to put the definition also in front of the type without unfolding it.

7.5. *Comparison with Automath*

Our item notation is influenced by the Automath notation (see [NGV94]). For example, de Bruijn uses the words *wagon*, *train* and *AT-pair* for item, segment and $\delta\lambda$-pair respectively. Furthermore, our definitions are also influenced by de Bruijn's introduction of definitions in his system AUT-$\Delta\Lambda$ (see B.7 of [NGV94]). In AUT-$\Delta\Lambda$, de Bruijn refines reduction in order to accommodate local reductions which are necessary for representing definitions. His main motivation in doing so are examples where the unfolding of a definition is usually desired at one specific instance of the definition and not everywhere it occurs.

Our presentation of definitions in this article, even though motivated by reasons similar to those of de Bruijn, is written in a way that fits elegantly with the style of the

cube. Our extension of the cube systems to deal with definitions only had to change the context a bit and to add the definition rule. De Bruijn claims Church Rosser for his mini reductions whereas we have studied all the cube systems with definitions and established their properties.

8. CONCLUSION

We have proposed an extension of β-reduction called generalised reduction and an extension of the typing rules of the cube with definitions. We use the item notation to make the generalised redexes clearly visible and to be able to describe nested definitions in a neat way.

Generalised reduction and definitions are shown to behave well with respect to β-reduction and the typing systems of the Barendregt cube.

With respect to reduction, generalised reduction has the Church–Rosser property and generates β-equality. SR has been studied and is shown to hold for all systems of the cube extended with definitions, which are also shown to be SN.

We showed that the extension of the typing systems, is a conservative one and that derivations become shorter without slowing down type-checking algorithms.

Before closing, it is worth mentioning where reductions related to our generalised notion have been used elsewhere. At the time of writing this paper, we were unaware of many related work and we are grateful to Joe Wells who has compiled most of the following details. We will be brief in what follows but we refer to [KW95b], which discusses the subject in detail.

Here are two rules related to our generalised reduction:

$$(\theta) \qquad (Q\delta)(P\delta)(\lambda_x)N \to (P\delta)(\lambda_x)(Q\delta)N$$

$$(\gamma) \qquad (P\delta)(\lambda_x)(\lambda_y)N \to (\lambda_y)(P\delta)(\lambda_x)N$$

It is obvious that θ may move the δ-item $(Q\delta)$ next to a λ-item in N if $N \equiv (\lambda_y)M$, and hence the δ-couple $(Q\delta)(\lambda_y)$ becomes a δ-pair making the generalised redex a classical one (visible) and subject to contraction. The rule γ is unrelated to what we do here yet has almost always been used with θ for technical reasons. (The transfer of rule γ to explicitly typed lambda calculus however, is not straightforward, since the type of y may be affected by the reducible pair $(P\delta)(\lambda_x)$.)

Regnier's notion of "premier redex" (see [Reg92]) is the same as our notion of generalised redex on untyped terms. We study it for Church-style type systems whereas Regnier studies Curry-style type systems. [Reg94] uses θ and γ (and calls the combination σ) to show that the perpetual reduction strategy finds the longest reduction path when the term is SN. [Vid89] also introduces reductions similar to those of [Reg94]. Furthermore, [KTU94] uses θ (and other reductions) to show that typability in ML is equivalent to acyclic semi-unification. [SF92] uses a reduction which has some common themes to θ. [dG93] uses a restricted version of θ and [KW95a] uses γ to reduce the problem of strong normalisation for β-reduction to the problem of weak normalisation for related reductions. [KW94] uses, amongst other things, θ and γ to reduce typability in the rank-2 restriction of system F to the problem of acyclic semi-unification. [AFM95] uses θ to analyse how to implement sharing in a real language interpreter in a way that directly corresponds to a formal calculus.

Received October 5, 1995; final manuscript received January 24, 1996

REFERENCES

[AFM95] Ariola, Z. M., Felleisen, M., Maraist, J., Odersky, M., and Wadler, P. (1995), A call by need lambda calculus, *in* "Conf. Rec. 22nd Ann. ACM Symp. Princ. Program. Lang.," Assoc. Comput. Mach., New York.

[Barendregt84] Barendregt, H. (1984), "Lambda Calculus: Its Syntax and Semantics," North-Holland, Amsterdam.

[Bar92] Barendregt, H. (1992), Lambda calculi with types, "Handbook of Logic in Computer Science," (S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, Eds.), Vol. II, Oxford Univ. Press, London/New York.

[BKKS87] Barendregt, H. P., Kennaway, J. R., Klop, J. W., and Sleep, M. R. (1987), Needed reduction and spine strategies for the λ-calculus, *Inform. and Comput.* **75**, 1191–1231.

[BKN94x] Bloo, R., Kamareddine, F., and Nederpelt, R. (1994), "Beyond β-Reduction in Church's $\lambda_{\to}$," Computing Science Note 94/20, Department of Mathematics and Computing Science, Eindhoven University of Technology.

[BKN94y] Bloo, R., Kamareddine, F., and Nederpelt, R. (1994), "The Barendregt Cube with Definitions and Generalised Reduction," Computing Science Note 94/34, Department of Mathematics and Computing Science, Eindhoven University of Technology.

[BKN9-] Bloo, R., Kamareddine, F., and Nederpelt, R. (submitted for publication), The λ-cube with classes which approximate reductional equivalence.

[CON86] Constable, R. L., *et al.* (1986), "Implementing Mathematics with the Nuprl Proof Development System," Prentice–Hall, New York.

[Dow91] Dowek, G., *et al.* (1991), "The Coq Proof Assistant Version 5.6, User's Guide," Rapport de recherche 134, INRIA, Le Chesney.

[Gardner94] Gardner, P. (1994), "Discovering Needed Reductions Using Type Theory," TACS.

[Geuvers94] Geuvers, H. (1994), A short and flexible proof of strong normalisation for the calculus of constructions, *in* "Types for Proofs and Programs," (P. Dybjer, B. Nordström, and J. Smith, Eds.), Selected Papers, International Workshop, Bastad, Sweden, Lecture Notes in Computer Science, Vol. 996, Springer-Verlag, Berlin/New York.

[GM93] Gordon, M. J. C., and Melham, T. F., Eds.), (1993), "Introduction to HOL: A Theorem Proving Environment for Higher Order Logic," Cambridge Univ. Press, London/New York.

[dG93] de Groote, P. (1993), The conservation theorem revisited, *in* "Int'l Conf., Typed Lambda Calculi and Applications," pp. 163–178, Lecture Notes in Computer Science, Vol. 664, Springer-Verlag, Berlin/New York.

[KN93] Kamareddine, F., and Nederpelt, R. P. (1993), On stepwise explicit substitution, *Int. J. Foundations Comput. Sci.* **4** (3), 197–240.

[KN94] Kamareddine, F., and Nederpelt, R. P. (1994), A unified approach to type theory through a refined λ-calculus, *Theoret. Comput. Sci.* **136**, 183–216.

[KN95] Kamareddine, F., and Nederpelt, R. P. (1995), Refining reduction in the λ-calculus, *J. Funct. Programming* **5**, No. 4, 637–651.

[KN96a] Kamareddine, F., and Nederpelt, R. P. (1996), A useful λ-notation, *Theoret. Comput. Sci.* **155**.

[KN96b] Kamareddine, F., and Nederpelt, R. P. (1996), Canonical typing and Π-conversion in the Barendregt cube, *J. Funct. Programming* **6**, No. 2.

[KTU94] Kfoury, A. J., Tiuryn, J., and Urzyczyn, P. (1994), An analysis of ML typability, *J. Assoc. Comput. Mach.* **41**(2), 368–398.

[KW94] Kfoury, A. J., and Wells, J. B. (1994), A direct algorithm for type inference in the rank-2 fragment of the second order λ-calculus, *in* "Proc. 1994 ACM Conf. LISP Funct. Program."

[KW95a] Kfoury, A. J., and Wells, J. B. (1995), New notions of reductions and non-semantic proofs of β-strong normalisation in typed λ-calculi, *in* "LICS '95."

[KW95b] Kfoury, A. J., and Wells, J. B. (1995), "Addendum to New Notions of Reduction and Non-semantic Proofs of β-Strong Normalisation in Typed λ-Calculi," Boston University.

[Launchbury93] Launchbury, J. (1993), A natural semantics of lazy evaluation, *in* "ACM POPL 93," pp. 144–154.

[Lévy80] Lévy, J.-J. (1980), Optimal reductions in the lambda calculus, *in* "To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism," (J. Seldin and R. Hindley, Eds.), Academic Press, San Diego.

[LP92] Luo, Z., and Pollack, R. (1992), "LEGO Proof Development System: User's Manual," Technical Report ECS-LFCS-92-211, LFCS, University of Edinburgh.

[Ned73] Nederpelt, R. P. (1973), "Strong Normalisation in a Typed Lambda Calculus with Lambda Structured Types," Ph.D. Thesis, Eindhoven, University of Technology.

[NGV94] Nederpelt, R. P., Geuvers, J. H., and de Vrijer, R. C., (Eds.) (1994), "Selected Papers on Automath," Studies in Logic and the Foundations of Mathematics, Vol. 133, North-Holland, Amsterdam.

[Reg92] Regnier, L. (1992), "Lambda calcul et réseaux," Thèse de doctorat de l'université Paris 7.

[Reg94] Regnier, L. (1994), Une équivalence sur les lambda termes, *Theoret. Comput. Sci.* **126**, 281–292.

[SF92] Sabry, A., and Felleisen, M. (1992), Reasoning about programs in continuation-passing style, *in* "Proc. 1992 ACM Conf. LISP Funct. Program," pp. 288–298.

[SP93] Severi, P., and Poll, E. (1993), "Pure Type Systems with Definitions," Computing Science Note 93/94, Eindhoven University of Technology, Department of Mathematics and Computing Science.

[Vid89] Vidal, D. (1989), "Nouvelles notions de réduction en lambda calcul," Thèse de doctorat, Université de Nancy 1.